

Dr. H.J.J. Uyttenhove

**PROGRAMMEREN
IN**

APL

SAMSOM

54,45

Programmeren in APL

Deze uitgave verschijnt in de reeks automatische informatie-
verwerking onder redactie van:

Drs. G.J. Groenenboom

Drs. Th. van Kooten

Prof. dr. ir. G.C. Nielen

Prof. J.M. van Oorschot

Prof. D. Steeman

Prof. dr. A.A. Verrijn Stuart

Dr. H.J.J. Uyttenhove

Programmeren in *APL*

Samsom Uitgeverij
Alphen aan den Rijn 1983

ISBN 90 14 03190 4
D/1983/0247/016

Copyright © 1983 H.J.J. Uyttenhove, Eindhoven

Behoudens de in of krachtens de Auteurswet van 1912 gestelde uitzonderingen mag niets uit deze uitgave worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke ander wijze dan ook, en evenmin in een retrieval system worden opgeslagen, zonder de voorafgaande schriftelijke toestemming van Samsom Uitgeverij bv, gevestigd te Alphen aan den Rijn, die daartoe door de auteursrechthebbende met uitsluiting van ieder ander is gemachtigd.

opdracht

aan Kris, Kim en Lina
en Eddie Fritz

Inhoud

Voorwoord	11
1 Inleiding	15
1.1 Computersystemen voor <i>APL</i>	16
1.1.1 <i>APL</i> op main-frame computers	17
1.1.2 <i>APL</i> op microcomputers	19
1.2 Het <i>APL</i> -toetsenbord	20
1.3 Enkele eenvoudige berekeningen	23
1.4 Alfnumerieke gegevens en variabelen	26
1.5 Organisatie van het <i>APL</i> -werkgeheugen	32
1.6 Invoerfouten en oefeningen	33
1.7 Oefeningen	35
2 Notaties voor gegevensverwerking	38
2.1 Scalairen en vectoren	38
2.2 Monadische en dyadische bewerkingen	39
2.3 Eenvoudige primitieve functies	41
2.3.1 Machtsverheffing	41
2.3.2 Afronding, minimum en maximum	42
2.3.3 Indiceren en getallen genereren	42
2.3.4 Elementenindicering	43
2.3.5 Toevalsgetallen	44
2.3.6 Absolute waarden en restwaarden	45
2.3.7 Lengte en vorm van vectoren	46
2.3.8 Samenvoegen en ontbinden	48
2.4 Logische functies	48
2.4.1 AND en OR functies	49
2.4.2 De functies $=$, $\neq$, $>$, $\geq$, $<$ en $\leq$	49
2.4.3 De ontkenningfuncties $\sim$, $\wedge$ en $\vee$	50
2.5 Oefeningen	51
3 Gedefinieerde functies en controle	54
3.1 Het definiëren van een functie in <i>APL</i>	54
3.2 Typen van gedefinieerde functies	58
3.3 Eenvoudige invoer en uitvoer	59
3.4 Verandering in een gedefinieerde functie	63
3.4.1 Toevoegen van regels	63

3.4.2	Weglaten van regels	65
3.4.3	Verbeteren van fouten in een regel	65
3.4.4	Veranderen van de functienaam	67
3.5	Toetsen van condities in functies	67
3.6	Lokale en globale variabelen	71
3.7	Gedefinieerde functies in functies	73
3.8	Oefeningen	77
4	Manipulatie van vectoren	80
4.1	Indiceren van vectoren	80
4.2	Splitsen van vectoren	82
4.3	Omdraaien van een vector	84
4.4	Het bepalen van de volgorde in vectoren	85
4.5	Coderen en decoderen van vectoren	86
4.6	Reduceren, onderdrukken en uitbreiden van vectoren	88
4.7	Vermenigvuldigen van vectoren	93
4.8	Oefeningen	95
5	Manipulaties van matrices	98
5.1	Definitie van een matrix	98
5.2	Indicering van een matrix	101
5.3	Opsplitsen van een matrix	102
5.4	Roteren en transponeren van een matrix	105
5.5	Sorteren in een matrix	108
5.6	Reduceren, onderdrukken en uitbreiden van een matrix	109
5.7	Vermenigvuldigen van matrices	110
5.8	Inverteren en delen van matrices	113
5.9	Oefeningen	114
6	Reeksen, logaritmen en goniometrische functies	117
6.1	Een gedefinieerde grafische functie	117
6.2	Combinatie- en faculteitsfuncties	124
6.3	De circelfunctie en logaritmen	125
6.4	Goniometrische functies	127
6.5	Oefeningen	130
7	Invoer- en uitvoerproblematiek	131
7.1	Invoer buiten een gedefinieerde functie	131
7.2	Invoer binnen een gedefinieerde functie	132
7.3	Uitvoer zonder formaat	134
7.4	Uitvoer met formaat	136
7.5	Samenvoeging van numerieke en alfanumerieke gegevens	139
7.6	Wijziging van uitvoer in een gedefinieerde functie	142
7.7	Oefeningen	144

8	<i>APL</i>-functies in de praktijk	145
8.1	Meer over gedefinieerde functies	145
8.1.1	Verzegelen van een functie	145
8.1.2	Verwijderen van regels binnen een functie	146
8.1.3	Variaties op verwijzingen binnen een functie	147
8.2	Het onderbreken en hervatten van een functie	148
8.2.1	Ongewenste onderbrekingen	148
8.2.2	Gewenste onderbrekingen	152
8.2.3	Resultaten per regel	154
8.3	Enkele eenvoudige hulpfuncties	155
8.3.1	Meervoudige instructies op een regel	155
8.3.2	Het testen van condities	157
8.3.3	Het tekenen van een histogram	158
8.4	<i>APL</i> -toepassingen voor de praktijk	160
8.4.1	Eenvoudige tekstverwerking	161
8.4.2	Huishoudelijke budgettering	166
8.4.3	Een statistische functie	172
8.4.4	Voorbeeld van computer ondersteund onderwijs (COO)	173
8.4.5	Voorbeeld van een financiële analyse	175
8.5	Oefeningen	181
9	Organiseren van het werkgeheugen en het <i>APL</i>-systeem	183
9.1	Het <i>APL</i> -systeem en het werkgeheugen	183
9.1.1	Manipulatie van het werkgeheugen	183
9.1.2	Communicatie met andere gebruikers	187
9.1.3	Publieke bibliotheken	188
9.1.4	Aansluiten op een <i>APL</i> -systeem	189
9.1.5	Afsluiten van de interactie	190
9.2	Organisatie van het werkgeheugen	190
9.2.1	Systeemcommando's	191
9.2.2	Systeemvariabelen	194
9.2.3	Systeemfuncties	198
9.2.4	Systeem-afhankelijke functies	201
9.3	Interactie met het computersysteem (shared variables)	202
10	Nabeschatting	204
	Appendix A: Foutmeldingen	208
	Appendix B: Beschikbare <i>APL</i>-systemen, 1983	212
	Register	215

Voorwoord

'A Programming Language' of *APL* is een computertaal gebaseerd op een notatie ontworpen door K. Iverson in 1957 en waarover in 1962 een eerste boek werd geschreven. Sedert de implementatie van *APL* op IBM-computers in de zestiger jaren, heeft deze taal toepassing gevonden in het onderwijs, bij onderzoek en in bedrijven. *APL* is een bondige programmeertaal waarmee hele reeksen van gegevens van verschillende aard met een beknopt aantal instructies door de computer kunnen worden verwerkt. Het elimineert vooral het werken in details, iets wat soms overweldigend is in andere computertalen.

Alhoewel *APL* lang beschouwd werd als een taal voor wiskundigen en technici, heeft het in de laatste tien jaar vaste voet gekregen in middelgrote tot grote bedrijven op grote computers en ook in kleinere bedrijven met de microcomputer. De behoefte aan een goed Nederlandstalig boek over *APL* kon dus niet uitblijven, zij het dan dat het in de eerste plaats een echt instructieboek geworden is om de taal aan te leren.

Dit *APL*-leerboek heeft als voornaamste doelstelling er voor te zorgen dat een lezer of student de *APL*-taal leert en dat het voor de diversiteit van toepassingen en de verdere bestudering van de taal gemakkelijk zal zijn dit boek als naslagwerk te hebben. De nadruk wordt gelegd op het leren van de basis *APL*-taal, dit wil zeggen die versie van *APL* die aansluit op de meeste bestaande *APL*-vertalers. De presentatie van de taal is doelbewust zo gekozen dat *APL* niet gebonden is aan bepaalde hardware. Zodoende kan dit boek worden beschouwd als inleidend en onafhankelijk van een of andere *APL*-versie van een leverancier.

Er is getracht de toepassingen voor zowel de student en de hobbyist als voor de applicatieprogrammeur op levendige en direct begrijpende wijze aan te geven. Mede door de indeling zal het boek in de toekomst als referentie kunnen dienen voor de lezer. Hiervoor zijn de nodige tabellen opgenomen in de appendices. Een belangrijk aspect van de taal en van de toepassing is de statistiek. Omdat het boek geschreven is voor een breed publiek, waaronder ook diegenen voor wie het werken met een computer nog onbekend is, zijn de wiskundige notaties over het algemeen niet benadrukt en alleen daar gebruikt waar het noodzakelijk is. In die gevallen waar een bepaalde vorm van de taal alleen maar van toepassing is in de wiskunde, wordt de lezer erop gewezen dat een stuk tekst en voorbeelden mag worden overgeslagen, zonder verlies aan basis informatie.

Het inleidend hoofdstuk schetst in het kort de omgeving waarin *APL* functioneert. Hierbij komen zowel microcomputers als grote computers aan bod. Pas als

de kennis van het voor *APL* kenmerkende toetsenbord vergaard is worden de eerste eenvoudige *APL*-instructies uitgevoerd. Dit komt neer op een bijgestuurde 'trial & error' methode zodat de lezer met een *APL*-terminal bij de hand ontdekt wat het effect van invoer is.

Het tweede hoofdstuk besteedt vooral aandacht aan het aanleren van notaties en aan de resultaten die het gebruik ervan opleveren. Hierbij worden zowel dyadische als monadische aspecten bekeken. *APL* functioneert hier nog buiten het eigenlijke programmeren om en komt overeen met een eenvoudige interactie van computer en mens, waar geen geprogrammeerde instructies naar de gebruiker of computer gestuurd worden.

Hoofdstuk drie gaat in op het voorbereiden van programma's in *APL* met daarbij de eenvoudige vormen van in- en uitvoer van gegevens. In *APL* noemen we programma's gedefiniëerde functies. Het veranderen van deze functies en de consequenties van het definiëren van lokale en globale variabelen spelen bij dit alles een grote rol.

Als voortzetting van hoofdstuk twee worden in het vierde hoofdstuk de speciale in de taal opgenomen functies voor scalaires en vectoren besproken. Daarbij worden voor elk van de geïntroduceerde functies voorbeelden als toelichting gegeven. Ook hier wordt het hoofdstuk afgerond met oefeningen.

Het onderwerp 'matrix' neemt het gehele vijfde hoofdstuk in beslag. Diezelfde functies welke werden toegepast op vectoren van één dimensie, krijgen nu een andere uitwerking als ze worden toegepast op vectoren van twee of meerdere dimensies die we matrices noemen.

Hoofdstuk zes bevat de functies en operatoren voor het verwerken van reeksen. Verder worden logaritmen en sinusfuncties beschreven. Dit hoofdstuk is van nature wat wiskundig en kan door de beginnende student worden overgeslagen, zonder verlies van continuïteit in het aanleren van de taal.

De meer geavanceerde in- en uitvoer mogelijkheden worden toegelicht in hoofdstuk zeven als vervolg op hoofdstuk drie. De hierin verwerkte stof is vooral geschikt voor *APL*-programmeurs die te maken krijgen met complexe in- en uitvoerformaten in het bedrijfsleven. Bij de uitvoerproblematiek komt vooral het door elkaar gebruiken van numerieke en alfanumerieke gegevens aan bod.

Hoofdstuk acht vormt een samenvoeging van de vergaarde kennis van de vorige hoofdstukken, aan de hand van enkele in detail uitgewerkte voorbeelden van *APL* in de praktijk. Hierbij komen speciale technieken en procedures voor, die het de *APL*-programmeur nog makkelijker kunnen maken.

Hoofdstuk negen bevat de relevante informatie voor het organiseren van een *APL*-werkgeheugen. Systeemcommando's, systeemvariabelen en systeemfuncties worden, voor zover deze bij de meeste *APL*-systemen voorkomen, toegelicht. Dit hoofdstuk wordt afgerond met die systeemfuncties die gebruikt worden bij het later in de *APL*-taal ontwikkelde concept van *gemeenschappelijke variabelen*.

Appendix A bevat een lijst van fouten die het *APL*-systeem meldt bij verkeerd gebruik, als ook de nodige uitleg en eventuele remedies ter verbetering van de fouten. Appendix B geeft een overzicht van de bij deze druk beschikbare *APL*-systemen.

Ten slotte wil ik graag diegenen te bedanken die met ijver het manuscript hebben doorgenomen en van commentaar hebben voorzien. Mijn dank gaat ook uit naar de Technische Hogeschool te Eindhoven en naar Digital Equipment b.v. te Utrecht voor het gebruik van de *APL*-faciliteiten.

1 Inleiding

De digitale computer is wellicht het krachtigste hulpmiddel dat ooit door de mens is ontworpen. Als zodanig is de 'computer revolutie' geweldiger en machtiger dan de agrarische, industriële en technologische revoluties die onze samenleving al heeft meegemaakt. Het is nog moeilijk de invloed van de computer objectief te schatten als het gaat om het verwerken en opslaan van gegevens bij bedrijven, overheidsinstanties en onderwijs.

Misschien is het minder voor de hand liggend, maar van een nóg groter belang is het effect hiervan op de manier waarop we problemen oplossen of laten oplossen. De computer heeft ons van vele taken ontlast, vooral die welke tijdrovend en wiskundig van aard zijn. Ook het opslaan en opvragen van gegevens via de computer is enorm toegenomen.

En toch, in het licht van al de verwezenlijkingen en verdere mogelijkheden van de computer, zijn er nog velen die er versteld van staan wanneer ze zich realiseren dat de machine zelf geen natuurlijke intelligentie heeft. Tot de weinige mogelijkheden van de computer behoren zeer eenvoudige bewerkingen zoals het optellen en aftrekken van twee getallen, het toetsen of een getal 0 of 1 is, en het elektronisch wegschrijven en lezen van deze getallen.

Om zelfs deze eenvoudige taken door een computer te laten doen moeten er heel wat instructies worden gegeven. Men kan op deze manier 'converseren' met de computer in *machinetaal*, een taal die kan worden opgevat als een verzameling van numerieke instructies. Het schrijven van het geheel van deze instructies heet *programmeren*.

Alhoewel het in het begin nog nodig was om op deze manier met de computer te converseren, is het reeds lang mogelijk één of andere computertaal te gebruiken die dicht bij de eigen natuurlijke taal staat. Bij sommige programmeertalen zijn de te gebruiken commando's op voorhand vastgelegd. Bij andere talen kan men een eigen woordenschat aanmaken, terwijl bij nog andere talen een combinatie mogelijk is.

De *APL*-taal is één van de meer elegante en moderne programmeertalen; zij is vooral krachtig in het oplossen van diverse problemen. Zij heeft het voordeel van een uitgebreide en krachtige notatie en een flexibele woordenschat, aangepast aan het terrein van het op te lossen probleem. Zowel deze taal als de vele andere talen moeten echter door de computer foutloos worden begrepen, dit wil zeggen de instructies moeten worden vertaald. Meestal gebeurt dit door een *compiler* die het programma in één keer omzet naar machinetaal. Het kan echter ook door een *interpreter* of vertaler gebeuren die alleen per uit te voeren instructie naar de

machinetaal omzet. *APL* is een voorbeeld van een vertalertaal die volgens enkele eenvoudige richtlijnen moet worden gebruikt binnen het bereik van het computersysteem.

Dit inleidend hoofdstuk geeft een indruk weer van wat de omgeving van *APL* zoal kan zijn, dus waar de gebruiker mee te maken krijgt, en hoe de computer met een taal als *APL* omgaat. Hierbij komen de meest voor de hand liggende zaken van computergebruik naar voren. Deze zijn echter ook belangrijk voor diegenen die reeds programmeren in een andere taal, omdat *APL* als vertaler een geheel andere instelling heeft ten opzichte van haar omgeving.

1.1 Computersystemen voor *APL*

APL beschrijven en aanleren gaat nu eenmaal niet zonder het gebruik van termen welke belangrijk zijn bij het gebruik van een computer. Die termen hangen soms af van de soort computer die wordt gebruikt. Hier en in de volgende hoofdstukken wordt er naar gestreefd algemeen te blijven en de taal centraal te laten staan.

Een computer heeft de mogelijkheid gegevens van verschillende aard op uiteenlopende manieren op te slaan. Dit boek bijvoorbeeld kan worden gezien als een zeer lange reeks karakters en speciale tekens die in een bepaalde volgorde voorkomen. Elk van deze karakters of tekens kan worden omgezet naar een code. Zo kunnen het karakter &, de letter Y of het teken ω omgezet worden in codes, respectievelijk 14, 22 of 101. De computer kan echter bij het lezen van de letter Y die code 22 onmiddellijk omzetten in een voor hem begrijpelijke code opgebouwd uit de cijfers 0 en 1. Dit omzetten naar een digitale code laat de computer toe dit gehele boek op te nemen en op een efficiënte manier op te slaan. Maar niet alleen tekst of een reeks getallen zijn vatbaar voor computerverwerking; ook formules, vergelijkingen en wiskundige notaties kunnen worden omgezet en verwerkt.

Om dit mogelijk te maken heeft de computer in hoofdzaak twee delen nodig. Deze delen zijn het best omschreven als *apparatuur* en *programmatuur*. Het eerste vormt de computer zelf met daarbij ook de machines die met de computer verbonden zijn. Dit is de randapparatuur, ook wel de periferie van de computer genoemd. Het tweede deel is het geheel van instructies aan de computer om zichzelf en de randapparatuur te sturen; dit zijn dan de programma's geschreven door een programmeur. Alle programma's bij elkaar kan men de programmatuur noemen. Bij *APL*-programmatuur spreekt men eerder over functies dan over programma's. In de Engelstalige literatuur heet dit dan *function*. Deze term is vanaf het ontstaan zo gekozen omdat de eerste geschreven programma's van *APL* zeer wiskundig waren georiënteerd.

APL kan momenteel op tientallen verschillende soorten computers worden gebruikt. Deze kunnen in twee groepen worden gesplitst volgens een verdeling welke globaal overeenkomt met de grootte van het computersysteem. Die grootte

is uitgedrukt in de capaciteit van het geheugen en de combinatie van mogelijkheden. Zo onderscheiden we de grote computers bekend als *main-frames* en de minicomputers aan de ene kant, en de kleine computers of de microcomputers aan de andere kant. Dit onderscheid maken we eigenlijk alleen maar in dit hoofdstuk omdat het over de apparatuur gaat. De *APL*-taal is voor ons belangrijker, en het maakt niet uit of men nu een grote of kleine computer tot zijn beschikking heeft.

1.1.1 *APL* op main-frame computers

Diverse rekencentra hebben één of meerdere grote computers waar de gebruiker terecht kan om in verschillende computertalen te programmeren. Aangezien *APL* meer en meer wordt toegepast, houdt dit in dat er ook op meer en meer computers *APL* geïnstalleerd wordt. Waar men vroeger deze installaties vooral bij hogere onderwijsinstellingen vond, kunnen we nu rustig stellen dat de gebruikers van *APL* nu zowel in het bedrijfsleven en bij de overheid als bij het onderwijs te vinden zijn.

De apparatuur waar *APL* mee te maken krijgt beantwoordt over het algemeen aan een combinatie van een reeks machines die aan de computer zijn gekoppeld. Daarbij onderscheiden we:

1. Een centrale verwerkingseenheid voor alle besturingsfuncties.
2. Een snel werkend relatief klein (intern) geheugen dat alle informatie zoals instructies en gegevens bevat die op een bepaald moment nodig zijn voor de centrale verwerkingseenheid.
3. Een trager werkend relatief groot (extern) geheugen dat de opgeslagen informatie bevat die niet meteen nodig is voor de verwerkingseenheid.
4. Een leeseenheid voor ponskaarten om gegevens en instructies in de machine in te voeren.
5. Een afdrukeenheid (printer) om gegevens, instructies of resultaten van verwerking van gegevens af te drukken.
6. Een controle-eenheid voor transmissie van gegevens via kabels en telefoonlijnen en dat toelaat meerdere terminals of randapparatuur aan de computer te koppelen.
7. Eén of meerdere terminals.

Deze opsomming is voor vele computercentra zeker niet uitputtend, maar voor de gebruiker van *APL* op een groot computersysteem is deze apparatuur ruimschoots voldoende, mits de gebruikte terminal een *APL* toetsenbord heeft (zie paragraaf 1.2).

Hoe werkt *APL* binnen zo'n main frame? Veronderstel dat de *APL* gebruiker via de terminal aan het computersysteem verbonden is, en dat na het invoeren van een bepaald nummer en een daarbij behorend wachtwoord (*password*) toegang gege-

ven wordt tot het eigenlijke gebruik van de computer. Een interactie verloopt als volgt. De gebruiker krijgt een hoeveelheid ruimte binnen het systeem toegewezen waarin kan worden gewerkt. Als er simultaan vele *APL*-programmeurs bezig zijn, dan blijft wel hetzelfde werkgeheugen, maar krijgt iedere gebruiker, volgens het concept van *time sharing*, minder verwerkingstijd. Dit kan bij bepaalde computersystemen wel eens tot problemen leiden. Deze problemen doen zich meestal voor in de vorm van lange wachttijden. De wachttijd is hier te verstaan als de tijd die iemand moet wachten op een antwoord van de computer op een gegeven instructie. Omdat *APL* nu eenmaal een vertaler is, lijkt dit nog erger omdat elke regel met een *APL*-instructie in een programma, bij ieder gebruik in machinetaal dient te worden omgezet, wat nog eens extra tijd vraagt. Moet eenzelfde regel door omstandigheden nogal dikwijls worden vertaald, dan zou dit een extra belasting van de verwerkingseenheid betekenen.

Als gevolg daarvan ziet men dat grote *APL*-gebruikers voor verwerking van bedrijfsgegevens gebruik maken van computers die ofwel meer door *APL* mogen worden belast dan door andere talen, ofwel volledig voor *APL* beschikbaar zijn. Andere gebruikers schakelen dan ook over op het meer individueel verwerken van de gegevens via een kleinere computer.

Die hoge belasting is echter geen algemeenheid en voor de meeste toepassingen voldoen de grote computers. Dit blijkt uit het feit dat de gemiddelde antwoordtijd van de computer niet hoger ligt dan één seconde. Daarbij komt nog dat het voordeliger is een grote computer te gebruiken omdat dan zowel het interne als het externe geheugen voldoende groot zijn voor groots opgezette onderzoeksactiviteiten en toepassingen.

De *APL*-programmeur maakt net zoals de programmeurs in andere talen, nu en dan gebruik van gegevens die staan opgeslagen in data banken, die op hun beurt te vinden zijn op magnetische banden of magnetische schijven. Deze vorm van opslag noemden we hiervoor bij punt 3 gemakshalve het externe geheugen. De mogelijkheid bestaat om gegevens, functies en resultaten van enige gegevensverwerking ook op andere randapparatuur te laten opslaan. Hiervoor wordt in *APL* het concept van gemeenschappelijke variabelen gebruikt zoals besproken in hoofdstuk negen.

Verder wordt door de *APL*-vertaler het eigenlijke geheugen van de computer gebruikt om er de actuele berekeningen of verwerkingen uit te voeren, en wordt er geen rekening gehouden met ponskaarten of de leeseenheid hiervoor. Deze laatste kunnen echter wel van pas komen buiten het *APL*-gebeuren om, bijvoorbeeld bij het inbrengen van gegevens.

Alhoewel de recente technologische ontwikkelingen het mogelijk maken om op individuele basis wat betreft *APL* met de microcomputer te gaan werken, blijft de combinatie van terminal en grote computer vooralsnog de meest gebruikte vorm.

1.1.2 APL op microcomputers

Reeds bij het beschrijven van het gebruik van APL op grotere systemen is het duidelijk gebleken dat er zich gebruikersonvriendelijke situaties kunnen voordoen. Dit is vooral relevant bij het lang uitblijven van een computerantwoord op een eenvoudige berekening. Er zijn echter nog een aantal andere redenen waarom men zou kunnen overgaan tot het gebruik van een microcomputer. Deze zijn: Het in eigen beheer houden van gegevens en de daarmee gemoeide *privacy*; verwerking onafhankelijk van anderen; verwerking niet gebonden aan onderhoudsbeurten bij grote computers; vrije keuze van waar en wanneer er moet verwerkt worden; lagere kosten bij aanschaf apparatuur; lagere kosten bij aanschaf APL-vertaler; enz..

Bij APL is er echter een bijkomende moeilijkheid in die zin dat de APL-vertaler een groot deel van het snelle of interne geheugen inneemt en zowel tijdens de verwerking als tijdens het programmeren in dat geheugen aanwezig dient te zijn. De naam van microcomputer impliceert reeds dat het een kleinere versie is van de grotere systemen, niet alleen de fysieke afmetingen, maar ook wat betreft het interne geheugen, de opslagcapaciteit, snelheid van de centrale verwerkingseenheid, enz..

Het interne geheugen moet minstens uit 64K bytes bestaan. Dit hoeft enige toelichting want bij de grotere computers staat de gebruiker niet stil bij de omvang van het geheugen. Bij de reeds eerder gemaakte opmerking dat dit boek voor verwerking vatbaar is, hebben we gesproken in termen van het coderen in 0 en 1. Die eenheid 0 of 1 noemt men in het computerjargon een *bit*. Zo'n bit komt eigenlijk overeen met de magnetische toestand op een bepaalde plaats in het geheugen. De computer kan deze toestand interpreteren als 'iets'(1) of 'niets'(0). Een bit kan derhalve slechts worden gebruikt om twee toestanden te onderscheiden, en is in wiskundige termen een binaire eenheid.

Met nog een bit erbij kan de computer reeds vier toestanden onderscheiden, namelijk de combinaties 00, 01, 10 en 11, zodat bij elke toevoeging van een bit er tweemaal zo veel toestanden kunnen worden onderscheiden als voorheen. Om dus alle cijfers, speciale tekens of APL-karakters, die in dit boek voorkomen op een unieke manier in het geheugen te kunnen beschrijven, is er een groep bits nodig per teken, cijfer en karakter. Deze groep van bits noemen we een *byte*. Afhankelijk van het type computer dat wordt gebruikt is het aantal bits in een byte groter of kleiner. Men drukt meestal het geheugen uit in het aantal bytes, of in duizenden bytes, afgekort K, en in het algemeen kunnen we stellen dat er per byte een teken kan worden opgeslagen. De opbouw van een byte met zeven bits zoals bij de ASCII-code, ziet er bijvoorbeeld voor de letter 'U' uit als 101010. Ook voor de instructies geldt dat de uiteindelijke machinecode in bytes wordt omgezet.

Bij een microcomputer met 64K bytes kunnen we in het algemeen ruim 64.000 bytes aan geheugen hebben. Als elke byte uit acht bits zou bestaan, dan komt dit

neer op een geheugen van rond een half miljoen bits. Ter vergelijking: Het volwassen menselijk brein van amper één kilogram bestaat uit ongeveer tien miljoen bits meestal supersnel werkend geheugen. De grootste computers hebben daarentegen een 'brein' van 128 miljoen bits, maar zijn niet zo efficiënt georganiseerd, en wegen ongeveer 100 kg, of slechts 1.28 miljoen bits per kilogram. Een microcomputer is dus heel klein maar kan toch bij het programmeren van steeds terugkerende berekeningen zeer efficiënt zijn.

Als men weet dat een gemiddelde *APL*-vertaler ongeveer 30K bytes geheugenruimte beslaat, en dat er ook nogal wat andere systeemprogrammatuur permanent aanwezig dient te zijn, dan is het duidelijk dat het werkgeheugen voor *APL* beperkt is tot minder dan 30K bytes. Voor bedrijfskundige toepassingen zou dit laatste tot problemen kunnen leiden bij het verwerken van grote hoeveelheden gegevens.

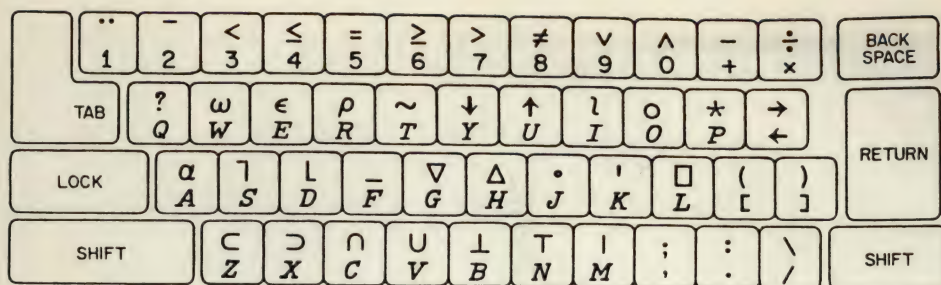
De apparatuur waarmee *APL* te maken krijgt bij de microcomputer is niet zo uitgebreid als bij een grote computer maar we kunnen toch de volgende delen onderscheiden:

1. Een centrale verwerkingseenheid voor alle besturingsfuncties.
2. Een snel werkend en klein intern geheugen.
3. Een groter extern geheugen om gegevens op te slaan.
4. Een toetsenbord met beeldscherm voor interactie.
5. Een kleine afdrukeenheid om gegevens, functies of resultaten af te drukken.

Uiteraard is deze lijst niet volledig en kan, afhankelijk van het type microcomputer, de apparatuur anders zijn. Als de *APL*-vertaler aanwezig is in het geheugen kan hij gebruikt worden via de centrale verwerkingseenheid. Wachttijden die te wijten zijn aan anderen doen zich hier niet voor. De eventuele beperkingen in de taal of opslagcapaciteit worden meestal door de *APL*-gebruiker geaccepteerd. Daar waar in de volgende hoofdstukken functies en dergelijke beschreven zijn en waar er voor microcomputer-gebruikers moeilijkheden zouden zijn zal dit in het kort toegelicht worden.

1.2 Het *APL*-toetsenbord

Zoals reeds vermeld bestaan er speciale *APL*-karakters die ook voorkomen op een speciaal hiervoor ontworpen toetsenbord. Op het eerste zicht lijkt dit toetsenbord op dat van een gewone elektrische schrijfmachine zoals in figuur hierna te zien is. Voor het alfabet zijn er cursief gedrukte hoofdletters overgebleven welke op dezelfde manier als kleine letters op de toetsen aangeslagen worden. Boven elk van die letters staat een speciaal *APL*-karakter dat meestal uit een wiskundige notatie is voortgekomen.



Verder is het toetsenbord zo ingedeeld dat de meest gebruikte tekens zich onderaan bevinden, en de minder gebruikte bovenaan. Ook valt op te merken dat een speciaal teken of karakter op die toets voorkomt van de letter of het teken waarmee het meest verwantschap wordt vertoond. Zo staan bijvoorbeeld A en α op één toets, R en ρ , $\leftarrow$ en $\rightarrow$, enz.

Omdat de controlettoetsen meestal in de Engelse taal worden weergegeven behoeft dit aan de hand van een paar voorbeelden enige uitleg.

Veronderstel dat men de apparatuur van OFF (uit) naar ON (aan) geschakeld heeft, dan kan men meteen *APL* gaan gebruiken in uitvoeringsmode, ook calculatormode of directe methode genoemd. Typt men bijvoorbeeld:

$$12 + 34$$

en daarna RETURN-(terugkeer)toets dan 'antwoordt' de computer met het resultaat van die optelling:

$$46$$

De SHIFT(verplaatsings)-toets wordt gebruikt om de tekens of karakters die zich bovenaan de toets bevinden te kunnen typen. Zo heeft:

$$19 \div 3$$

na de SHIFT-toets gebruikt te hebben voor het deelteken $\div$, en de RETURN-toets, als gevolg dat de computer reageert met:

$$6.333333333$$

Heeft men echter een fout gemaakt en zou het niet 19 maar 15 moeten zijn, dat door 3 moet worden gedeeld, dan kan men alsnog de cursor bij een beeldscherm of de typebol bij een schrijfmachine-terminal terug naar links bewegen, vóór het indrukken van de RETURN-toets. Daarvoor wordt de BACKSPACE toets gebruikt. Hiermee gaat men terug tot de 9 , waarna de ATTN-toets kan worden ingedrukt. ATTN is de afkorting van ATTENTION en laat de invoer vanaf dat

Op het toetsenbord is de gehele reeks van tekens en *APL*-karakters overzichtelijk te vinden. Wel moet er op worden gelet de letter *O*, de nul (0) en het teken $\circ$ niet met elkaar te verwarren. Deze opmerking geldt ook voor datzelfde teken $\circ$ en het teken $^{\circ}$ dat zich boven de letter *J* bevindt. Het verschil tussen de tekens $\sim$ en $\sim$ die respectievelijk links en rechts bovenaan staan zal nog verder worden toege-licht.

1.3 Enkele eenvoudige berekeningen

Zonder van echt programmeren te spreken weten we reeds dat het mogelijk is de computer met *APL* als een calculator of in uitvoeringsmode te gebruiken. We weten ook dat de *APL*-programmeur de computer bepaalde bewerkingen wil laten verrichten die veelal op rekenen zullen neerkomen. Voor diegenen die een *APL*-terminal of computer bij de hand hebben is het gemakkelijk deze berekeningen op deze manier te testen.

Optellen, aftrekken, vermenigvuldigen en delen zijn gemakkelijk te doen, net zoals op een calculator. Men gebruikt hiervoor de gewenste getallen die gescheiden zijn door de tekens $+$, $-$, $\times$ of $\div$. Na elke invoer van de berekening wordt de terugkeertoets gebruikt, zodat men bij de volgende voorbeelden de daarbij passende resultaten krijgt te zien:

```

              441+559
1000
              747 - 319
428
              421×108
45468
              19481 ÷ 17
1145.941176

```

Hierbij zal de lezer gemerkt hebben dat al hetgeen hijzelf invoert iets verder op de regel gebeurt omdat daar de typebol van de schrijfmachine of de cursor van het beeldscherm staat. Dit gebeurt bij de meeste *APL*-systemen automatisch. Het antwoord van de computer begint dan altijd op de volgende regel zes posities naar links, tenzij dit door een instructie anders wordt aangegeven.

De bovenstaande voorbeelden zijn natuurlijk heel gemakkelijk te herhalen met andere getallen. Probeer ook eens getallen met decimalen te gebruiken zoals:

```

              217.14 ÷ 4.06
53.48275862

```

De resultaten worden altijd weergegeven tot het laatst berekende significante cijfer. Het totaal aantal cijfers, inclusief een komma (een punt in de Engelse schrijfwijze!) is voor de meeste *APL*-systemen gebonden aan een maximum. Zonder wijziging van de gebruiker is dit maximum acht cijfers.

Men kan verschillende berekeningen op één regel uitvoeren door meerdere getallen in te voeren en die telkens te scheiden door het gewenste teken. Zo kan men berekenen uit hoeveel bits een centrale verwerkingseenheid van 48K bestaat, gegeven dat K gelijk is aan 1024 bytes en dat elke byte bestaat uit acht bits, of men berekent hoeveel belasting er moet worden betaald op een inkomen van 45.250 gulden als er op de eerste 30.000 gulden 7.000 gulden moeten worden betaald, en 40 procent over het resterende bedrag. De resultaten van beide berekeningen kunnen als volgt worden gevraagd:

$$48 \times 1024 \times 8$$

$$393216$$

Bij deze laatste berekening is enige toelichting nodig. Diegenen die in andere computertalen geprogrammeerd hebben zullen gemerkt hebben dat er geen haakjes gebruikt zijn om de verschillende berekeningen op de juiste getallen te laten toepassen. Ook zal men wellicht denken dat er een hiërarchische volgorde tussen de tekens bestaat. Dit is echter in *APL* niet het geval en daarom dient de regel voor de berekening van de belasting van RECHTS naar LINKS te worden gelezen. Dat geldt voor elke *APL* instructie, dus ook als er gebruik wordt gemaakt van haakjes.

De berekening dient dus als volgt te worden gelezen: 30.000 wordt afgetrokken van 45.250, hetgeen resulteert in 15.200. Dit laatste getal wordt vermenigvuldigd met .40 wat 6.100 geeft. De som van 6.100 en de gegeven 7.000 is dus het eindresultaat 13.100. Om hetzelfde resultaat te bereiken kan men echter, door gebruik te maken van haakjes, ook schrijven:

$$(7000 + (.4 \times (45250 - 30000)))$$

$$13100$$

In verschillende *APL*-tekstboeken kan men nog een actueel onderwerp vinden dat het gebruik van de rechts-naar-links regel goed aantoonst. Naar een Amerikaanse onderzoeker beweert, heeft een 50 jaar oude persoon die vanaf de leeftijd van 21 jaar elke dag een pakje sigaretten rookt, zijn leven verkort met ongeveer 8.5 jaar. Als we nu aannemen dat het precies één pakje van twintig sigaretten per dag is, dan kunnen we berekenen hoeveel minuten zijn leven per sigaret wordt verkort. Om het aantal minuten in een jaar te berekenen schrijven we:

$$365 \times 24 \times 60$$

$$525600$$

Dit resultaat kunnen we in de totale berekening gebruiken door het totaal aantal minuten in de 8.5 jaar te delen door het totaal aantal gerookte sigaretten in de genoemde periode, of als:

$$\frac{525600 \times 8.5 \div 20 \times 365 \times 50}{21.10344828} - 21$$

In het algemeen blijkt het dus dat, alhoewel de rechts naar links regel altijd van kracht is, elke functie zoals aftrekken of vermenigvuldigen de berekening uitvoert op de waarde onmiddellijk links van de functie, en de waarde van de *totale* instructie rechts van de functie.

De vorm van presentatie van de gegevens is net zoals bij andere programmeertalen ofwel zoals men deze dagelijks gebruikt. Maar ook zeer kleine en zeer grote getallen kunnen in *APL* worden weergegeven, zij het dan op een meer efficiënte manier. Zo is bijvoorbeeld de afstand die het licht in één jaar tijd aflegt, bij een snelheid van 300.000 km per seconde, een heel groot getal. Dat is ongeveer 9.500.000.000.000 km. Ook in *APL* worden getallen meestal in exponentiële vorm uitgedrukt als ze meer dan een bepaald aantal cijfers beslaan. Deze exponentiële vorm is aangegeven door de letter *E*, gevolgd door een getal. Ook de letter *E* zelf wordt voorafgegaan door een getal. Zo is het aantal km gegeven als *9.5E12* te interpreteren als 9.5 km maal 10 tot de macht 12. Met andere woorden, het eerste getal geeft de significante cijfers aan, de letter *E* staat voor 'maal 10 tot de macht' en het laatste getal is de macht.

Voor kleine getallen zoals .0000001 wordt dezelfde exponentiële vorm gebruikt als *1.0E-7*. Hier is het getal 1 tot de (negatieve) macht -7 verheven wat neer komt op het delen van 1.0 door een 1 met zeven nullen. Het gebruik van het teken - is dus verschillend van het minteken - dat als functie dient. Het negatief teken - heeft dus geen enkele functie en dient alleen maar om aan te geven dat het om een negatief getal gaat.

Gegevens kunnen ook in deze exponentiële vorm worden ingevoerd, als het om niet extreem kleine of grote getallen gaat. Zo zijn de getallen *1.234E3* (1234 dus) en *6.2E-2* (0.062) ook toegelaten. Let er echter wel op dat de letter *E* altijd aansluit op het vorige getal, en dat het onmiddellijk gevolgd wordt door de machtsverheffing, dus zonder spaties. Uiteraard dienen duizendtallen, indien geen gebruik gemaakt van een exponentiële vorm, zonder de punt te worden ingevoerd. Dus 43.456 wordt ingevoerd als *43456*, omdat er met een punt het getal 43,456 zou staan. Staat er toch een punt in het getal dan moet dat vanzelfsprekend een decimale punt zijn.

Tot nu toe hebben we *APL* gebruikt als calculator en is het gebleken dat dit zeer eenvoudig is. Elke keer werden er twee getallen opgeteld, afgetrokken, vermenigvuldigd of gedeeld. Veronderstel nu dat er het volgende probleem moet worden opgelost: In een restaurant heeft men een menu waar de hoofdschotels in vijf verschillende prijsklassen vallen. Deze vijf prijzen zijn 19.95, 22.95, 26.95, 29.95

en 34.95 gulden. Als de manager van het restaurant zou weten dat iedere schotel ongeacht de prijs voor het restaurant een winst van 40 procent betekent, dan zou hij de echte kosten per prijsklasse kunnen vaststellen door elke prijs met .6 te vermenigvuldigen. Dit kan men met *APL* zoals in de vorige voorbeelden doen. Dit betekent dat er vijf maal een prijs met .6 dient te worden vermenigvuldigd. Wat zou er nu gebeuren als we eens alle prijzen in één keer vermenigvuldigen met .6? Dat gebeurt als volgt:

$$\begin{array}{ccccccccc} & .6 \times & 19.95 & 22.95 & 26.95 & 29.95 & 34.95 & & \\ 11.97 & 13.77 & 16.17 & 17.97 & 20.97 & & & & \end{array}$$

Het resultaat bestaat nu uit vijf getallen die ieder de kostprijs per prijsklasse weergeven! *APL* heeft alle prijzen vermenigvuldigd met .6 omdat deze prijzen als een *vector* of een reeks werd opgegeven. Men ziet dus dat de bewerking zowel op individuele getallen als op een verzameling die we een vector noemen van toepassing is.

Om dit concept nog wat verder uit te breiden stellen we nu dat de kostprijzen voor iedere prijsklasse verschillende percentages hebben. Deze percentuele kosten zijn respectievelijk 50, 60, 60, 75 en 80 procent. Om nu uit te rekenen wat die kosten per prijsklasse zijn, passen we bij de percentage-getallen hetzelfde vectorprincipe toe:

$$\begin{array}{ccccccccccc} .5 & .6 & .6 & .75 & .75 & \times & 19.95 & 22.95 & 26.95 & 29.95 & \\ 34.95 & 9.975 & 13.77 & 16.17 & 22.4625 & 26.2125 & & & & & \end{array}$$

Opnieuw is het antwoord in de vorm van vijf getallen waar elke waarde correspondeert met één enkele prijsklasse, dus 9.975 is 50% van 19.95, enz. Het resultaat wordt dus verkregen door vermenigvuldiging van de respectievelijke componenten van de twee vectoren. Deze vectoren zijn numeriek en elk getal dient te worden gescheiden van een vorig getal door één of meerdere spaties.

Het gebruik van vectoren is dus heel handig en een stuk krachtiger dan met een gewone calculator, omdat een probleem als het vermenigvuldigen van twee vectoren of een vector met een getal, hetzelfde is als het vermenigvuldigen van twee getallen. Het gebruik van het woord vector moet de lezer hier niet meteen afschrikken omdat het een wiskundige term is. Naar willekeur kan men de term 'reeks van getallen' gebruiken als het om een numerieke vector gaat zoals hierboven in de voorbeelden.

1.4 Alf numerieke gegevens en variabelen

Naast de 'reeks van getallen' kunnen we onder *APL* ook een 'reeks van karakters' hebben. Alf numerieke gegevens zijn gegevens die bestaan uit zowel karakters en numerieke gegevens waarbij deze laatste echter ook als karakters dienen te worden geïnterpreteerd. Dus, het cijfer 8 is verschillend van het karakter '8'.

Als we even teruggaan naar het voorbeeld van het restaurant, dan hebben we vijf prijsklassen. Elke prijsklasse kunnen we een alfabetische code geven, bijvoorbeeld de letters A, B, C, D en E. Daarbij komt A overeen met de laagste prijs, enz. Deze codes kunnen in een vector worden geschreven als 'ABCDE', dus tussen aanhalingstekens en zonder spaties tussen de letters. Dit heet in *APL* een karaktervector en bestaat uit vijf elementen, namelijk de afzonderlijke letters. Het is zeer belangrijk te onthouden dat elke positie binnen de aanhalingstekens een element betekent, ook al staat daar een spatie. Als we in de karaktervector tussen alle letters een spatie laten, echter niet vóór *A* en niet na *E*, dan bestaat de vector uit negen elementen, te weten vijf letters en vier spaties.

Een karakter vector zoals '12345' kan echter *niet* als een getal worden geïnterpreteerd, althans niet zonder een ingreep van een programma. Dit blijkt uit onderstaande berekening als men deze vector zou gaan vermenigvuldigen met

.7

```

      .7 × '1234'
DOMAIN ERROR
0.7 × '1234'
^

```

Hier krijgt men een foutmelding waaruit blijkt dat de vermenigvuldiging niet is toegelaten op een karaktervector.

Karaktervectoren worden gebruikt voor alfanumerieke informatie zoals naam, leeftijd, salaris, woonplaats, enz. Zolang al deze informatie zich tussen de aanhalingstekens bevindt kan men er geen berekeningen mee uitvoeren, maar men kan het wél gebruiken voor uitvoer, of om bepaalde resultaten van enig commentaar te voorzien. Karaktervectoren zullen bij de problematiek van de uitvoer nog verder worden besproken.

De informatie die zich in vectoren bevindt is echter meer van numerieke aard zoals de prijsklassen van de schotels of het menu. Om nu nog een stapje verder weg te gaan van het als calculator gebruiken van *APL*, bekijken we hoe we die reeks van prijzen bijvoorbeeld onder één kenmerk kunnen opslaan. Het is immers vervelend, als een berekening met die vijf prijzen moet worden gemaakt, ze telkens opnieuw te typen, met het risico dat men fouten maakt. Om dit voor elkaar te krijgen gebruiken we *variabelen*.

Een variabele kan het best worden omschreven als een naam die een waarde heeft. De reden dat we het een variabele noemen komt voort uit de 'variërende' eigenschap van die waarde. Zo kan een variabele *X* een waarde van 200 hebben, of een waarde van 100, afhankelijk van bepaalde omstandigheden. Die '*X*' is een naam voor de variabele, en telkens als we deze naam gebruiken, gebruiken we eigenlijk de waarde die er op dat moment mee verbonden is. Als tegenhanger van de variabele hebben we de constante die altijd dezelfde waarde heeft. Zo is het getal 3 altijd 3.

Als voorbeeld voor een variabele, kiezen we de naam *PRIJS* voor de verschillende prijzen van de schotels. Om de prijzen aan deze naam te koppelen, gebruiken we in *APL* een speciaal teken dat we het toewijzingsteken noemen. Het is het horizontale pijltje naar links ($\leftarrow$) en het bepaalt dat de waarde(n) rechts ervan moet(en) worden opgeslagen in de variabelenaam links van de pijl. Voor de prijsklassen gebeurt dit dus als volgt:

PRIJS ← 19.95 22.95 26.95 29.95 34.95

De variable *PRIJS* bevat nu de vijf prijzen en kan ook als een numerieke vector worden beschouwd. Dit wil zeggen dat elke berekening die met deze variabele wordt uitgevoerd, overeenkomt met de berekening op alle getallen in deze vector, zodat de 60 procent berekening van voorheen nu heel eenvoudig neerkomt op:

$.7 \times \textit{PRIJS}$
13.965 16.065 18.865 20.965 24.465

Dit maakt het heel wat gemakkelijker voor de restauranthouder om, als het percentage van .6 naar .7 gaat, de berekening even eenvoudig te laten uitvoeren. Er is echter een probleem met de variabele die de prijzen bevat. Als de interactie met de computer via *APL* is gedaan en de machine wordt uitgezet, dan wordt ook het 'geheugen' uitgewist en zullen bij een volgende keer de prijzen opnieuw moeten worden ingevoerd. Om het 'verlies' van de prijsgegevens te voorkomen kan men in *APL* gebruik maken van een *SAVE* kommando waardoor al de gegevens en resultaten voor een volgende keer bewaard blijven. In het volgende deel van dit hoofdstuk gaan we hier verder op in.

De variabelen moeten ook nog aan een paar voorwaarden voldoen. Ten eerste moet elke variabele-naam uit één enkel woord bestaan. Ten tweede moet een naam beginnen met een letter van het alfabet of een letter van het alfabet onderlijnd met $_$, of het delta teken Δ met of zonder onderlijning. De maximum lengte van de naam van de variabele is voor de meeste *APL*-vertalers op 77 karakters gesteld. Indien variabelenamen langer zijn dan het toegestane aantal karakters, wordt de rest door het systeem genegeerd. Hierna volgt een lijst van goede en foutieve namen van variabelen:

GOED

TIJD

B

VAR01

RESULTAAT

EIND Δ TOTAAL

TESTD $_$ 1

FOUT

4DE

VARIABELE EEN

EIND BEDRAG

SNELHEID 1

14

$_$ 1NU

De toewijzing van waarden naar variabelen kan op meerdere manieren gebeuren. Een paar variabelen kunnen dezelfde waarden hebben en worden dan op dezelfde regels toegewezen als:

$$A \leftarrow B \leftarrow 1 \quad 0$$

Dit resulteert in de waarden 1 0 in de beide vectoren A en B . Men kan trouwens altijd aan de computer vragen wat een variabele bevat door gewoon de naam van de variabele in te voeren zonder verdere instructie. Willen we echt weten wat A en B zijn, dan doen we dit met:

$$\begin{array}{rcl} & & A \\ 1 & 0 & \\ & & B \\ 1 & 0 & \end{array}$$

De toewijzingspijl is in *APL* ook een functie zodat hij net als het plusteken ook verschillende keren op dezelfde regel kan voorkomen, zoals in:

$$\begin{array}{l} X \leftarrow 4 + Z \leftarrow 1 \\ C \leftarrow 6 + Q \leftarrow 4 \quad 14 \\ P \leftarrow 1 \\ P \leftarrow P + 1 \end{array}$$

De waarden van de variabelen X , P en C zijn nu als volgt

$$\begin{array}{rcl} & & X \\ 5 & & \\ & & P \\ 2 & & \\ & & C \\ 10 & 20 & \end{array}$$

Vasthoudende aan de rechts-naar-links regel, heeft *APL* eerst de waarde 1 toegekend aan variabele Z , het dan bij de waarde 4 geteld, en vervolgens dit resultaat aan de variabele A gegeven. In het tweede voorbeeld werd de variabele Q als een vector gespecificeerd en kreeg de waarden 4 en 14. Het plusteken opereert op de gehele vector met het getal 6 zodat het resultaat $(6+4) (6+14)$ aan variabele C wordt gegeven. De waarde van de variabele P is bij de eerste toewijzing 1, en bij de daarop volgende regel wordt de waarde 1 bij de de waarde van P geteld, dus 2, en opnieuw aan variabele P gegeven. Dit laatste geval is een voorbeeld van respecificatie van een variabele en komt vooral in programma's voor die met tellers moeten werken.

Voor het volgende onderwerp keren we terug naar de prijsklassen in het restaurant. De eigenaar besluit dat het menu aan uitbreiding toe is. Dit houdt in dat er een serie kindermenu's bijkomen van 14,95 en een schotel voor twee personen van 64,95 gulden. Om deze twee getallen aan de bestaande vector *PRIJS* toe te voegen kunnen we gebruik maken van een zeer eenvoudige *APL*-functie, namelijk de komma als samenvoegingsteken. Eigenlijk is het gebruik van de komma niet zo vreemd. Zo zouden bij de toewijzing van de vector, de vijf getallen kunnen worden ingevoerd, gescheiden door een komma. *APL* zou dit dan hebben geïnterpreteerd als een samenvoeging van de verschillende getallen tot een vector. Deze methode van samenvoegen van gegevens wordt ook wel *lijmen* genoemd. Bij de aanpassing van de variabele door de nieuwe prijzen, en gebruik makende van de respecifiëcatie, voeren we nu het volgende in:

PRIJS ← 14.95 , *PRIJS* , 64.95

De variabele met de prijsklassen bevat nu zeven in plaats van vijf elementen, waarbij 14,95 het eerste en 64,95 het laatste getal is.

Het bovenstaande voorbeeld van een samenvoeging illustreert reeds het samenvoegen van meer dan een getal, een variabele en een vector. De oefeningen aan het eind van dit hoofdstuk kunnen deze techniek nog verder toelichten voor de lezer.

Als tweede voorbeeld nemen we een eenvoudige berekening zoals:

((6+7) × .4) , 2.6 4.3
5.2 2.6 4.3

Bij dit voorbeeld speelt ook de volgorde van uitvoering, dus van rechts naar links, een rol. De komma vóór het getal 2.6 voegt de vector (2.6 4.3) toe aan de waarde van links. Omdat daar de prioriteit van uitvoeren volgens de haakjes verloopt, wordt .4 vermenigvuldigd met 13 (de som van 6 en 7) met als resultaat 5.2 of het element in de vector waaraan de andere getallen moeten worden toegevoegd. De betekenis van het woord toevoegen is hier dus niet optellen maar wel het naast elkaar zetten van getallen.

Omdat de komma bij *APL* als een functie wordt gebruikt moet er altijd een getal, teken of variabele links en rechts van de komma voorkomen. Voor Europese gebruikers heeft dit soms een overwacht resultaat vooral bij het invoeren van getallen. Dit is bijvoorbeeld hieronder het geval:

K ← 1, 2 3, 4 5, 6
K
1 2 3 4 5 6
Q ← 1.2 3.4 5.6
Q
1.2 3.4 5.6

Voor al in het begin voert men de toewijzing van decimale getallen uit zoals dat hierboven bij *R* gebeurt. *APL* interpreteert dit als een vector van driemaal een samenvoeging van twee getallen, zodat de uiteindelijke vector uit zes getallen bestaat. De invoer moet gebeuren via de Angelsaksische methode waar punten de komma's vervangen. Bij *Q* is hierboven wel de gewenste vector ingevoerd.

Voor karaktervectoren gelden dezelfde regels. De samenvoeging gebeurt echter uitsluitend met karaktervectoren, net zoals de samenvoeging van getallen met getallen moet gebeuren. In het algemeen zeggen we dat de variabelen of invoerelementen van hetzelfde type moeten zijn.

Als we bijvoorbeeld de twee woorden '*APL*' en '*TAAL*' willen samenvoegen met een spatie ertussen dan kunnen we schrijven:

```
'APL',' ','TAAL'
APL-TAAL
```

De drie vectoren worden samengevoegd tot één enkele vector. Indien de komma's niet worden gegeven dan antwoordt de computer met een foutmelding. Bij het begin van de oefeningen worden de mogelijke fouten aan de hand van voorbeelden toegelicht.

We kunnen nu ook een verhaaltje gaan schrijven aan de hand van enkele variabelen, die ieder een vector met als inhoud een woord voorstellen. Enige combinaties leiden dan tot de volgende zinnen:

```
D←'DE MAN '
F←'FRANS '
E←'EN '
H←'HEET '
S←'SPREEKT '
```

```
D,S,F
DE MAN SPREEKT FRANS
```

```
D,H,F
DE MAN HEET FRANS
```

```
D,S,F,E,H,F
DE MAN SPREEKT FRANS EN HEET FRANS
```

```
S,E,H,D,F,'?'
SPREEKT EN HEET DE MAN FRANS ?
```

De lezer kan zelf aan de hand van de oefeningen een paar taalkundige toepassingen uitwerken.

1.5 Organisatie van het *APL*-werkgeheugen

Om dit inleidend hoofdstuk af te sluiten keren we even terug naar de omgeving van *APL*. Vooral belangrijk is het gebruik van grotere interne en externe geheugen.

Bij de bespreking van de grootte van een *APL*-werkgeheugen op microcomputers hebben we gezien dat er een bepaald aantal bytes nodig is voor het werken met *APL*. Dit werkgeheugen heet bij *APL* de *WORKSPACE*, en bij het begin van een interactieve sessie met *APL* wordt een ruimte in het geheugen voor de gebruiker beschikbaar gesteld. Al het werk, in de vorm van programmeren of toewijzen van gegevens aan variabelen, blijft voorlopig opgeslagen in dit werkgeheugen. Dit geheugen wordt immers bij het uitzetten van het toetsenbord niet automatisch opgeslagen op bijvoorbeeld een magnetische schijf. Bij de eenvoudige voorbeelden die we in dit hoofdstuk hebben gezien is dit niet zo erg omdat er slechts enkele variabelen werden gebruikt. Zouden we echter programma's hebben geschreven, dan was de zaak wellicht anders geweest.

Het voorlopige werkgeheugen kan met een naam geïdentificeerd worden. Pas als die naam vaststaat kan men onder die naam het gedane werk opslaan in het externe geheugen. Dan bestaat ook meteen de mogelijkheid het geheugen weer actief te maken door het werkgeheugen onder de gegeven naam weer op te roepen naar het interne geheugen van de computer. Dit gebeurt via commando's die voor zichzelf spreken. Achtereenvolgens zien we hieronder het geven van een naam aan het werkgeheugen, het opslaan van dit geheugen, en het oproepen ervan:

```

)WSID INLEIDING
WAS CLEAR WS
)SAVE
SAVED 15:22:10 10/10/81 INLEIDING
)LOAD INLEIDING
SAVED 15:22:10 10/10/81

```

Hierbij is het sluithaakje `)` het begin van een kommando in *APL* om met het externe geheugen in verbinding te treden. De naam van het werkgeheugen is *WSID*; deze is willekeurig gekozen.

Alhoewel de lezer zich op dit moment nog niet hoeft af te vragen wat er allemaal achter de schermen gebeurt, is het toch nuttig enkele commando's nu reeds te gebruiken. Ze zijn namelijk zeer belangrijk als men van het gepresteerde werk nog iets wil bewaren. Hiermee verband houdend staat het commando *COPY* dat toelaat uit een reeds bestaand en opgeslagen werkgeheugen iets te kopiëren naar het actief werkgeheugen. Een *APL*-gebruiker kan een hele reeks van werkgeheugen hebben opgeslagen afhankelijk van de grootte van de computer of de mogelijkheid tot het verwisselen van de kleine magnetische schijfjes (*floppy disks*) bij microcomputers.

Laten we even aannemen dat we momenteel in een werkgeheugen aan het werk zijn die *TEST* heet, en dat we de variable *A* uit het werkgeheugen *INLEIDING* nodig hebben. Deze variabele kunnen we copiereën door het commando als volgt te gebruiken:

```

)COPY INLEIDING A
SAVED 15:22:10 10/10/81
A
1 0
```

Als variatie hierop kan men een reeks van variabelen en functionele programma's overschrijven, of gebruik maken van bestaande werkgeheugens die voor alle gebruikers beschikbaar zijn in een *APL-bibliotheek*. Een bibliotheek is een verzameling van werkgeheugens opgeslagen in het externe geheugen waarvan per commando, één werkgeheugen of gedeelten daaruit in het actief werkgeheugen kunnen worden gebracht. Zo'n bibliotheek kan op bepaalde grote computers soms zeer uitgebreid zijn, vooral bij computers die voor 100 procent voor *APL* worden gebruikt. Een laatste variatie is het kopiëren van informatie uit een werkgeheugen van iemand anders. Wachtwoorden om toegang te krijgen zullen hier wellicht nodig zijn maar dit hangt weer af van het type computer en de *APL*-vertaler.

De organisatie binnen het werkgeheugen zelf wordt bepaald door het type *APL* dat op een bepaalde computer is geïmplementeerd. Alhoewel dit per computersysteem en leverancier kan verschillen zal er voor de gebruiker over het algemeen geen verschil merkbaar zijn. Er is in elk werkgeheugen plaats voorbehouden voor een aantal variabelen, voor een aantal programma's, alsook voor reeds ingebouwde systeemvariabelen. Van deze laatste zijn er enkele die door de *APL*-gebruiker kunnen worden veranderd. In dit boek wordt dit waar het van belang is, uitvoerig toegelicht.

1.6 Invoerfouten en oefeningen

APL werkt als een vertaler en kan daardoor bij het lezen van een instructie of commando meteen zien of er een fout is gemaakt. Het kan soms ook gebeuren dat er op de transmissielijn naar een grote computer of op een kabel van het toetsenbord naar de verwerkingseenheid storing optreedt. In die gevallen antwoordt het systeem met:

RESEND

waarna de ingevoerde regel opnieuw moet worden ingetypt. Ook al is die regel goed bij de *APL*-vertaler angekommen, dan kan het nog zijn dat de gebruiker zelf één of andere fout heeft gemaakt. Als deze fout tegen de regels van de *APL*-taal gemaakt is, dan zal de vertaler deze fout melden. Bij logische fouten, zoals bij het

programmeren wel eens kan voorkomen, zal de vertaler geen oordeel vellen en worden geen fouten gemeld. Dus bij syntactische fouten zoals bijvoorbeeld het vergeten van een openend of sluitend haakje, antwoordt de computer met:

```

A←←14+2
SYNTAX ERROR
A←←14+2
^

```

Als er bijvoorbeeld voor een variabele een naam wordt gebruikt waarvoor nog geen toewijzing werd gegeven, antwoordt de computer met een foutmelding. Als de letter *X* zou zijn gebruikt in plaats van het teken $\times$, dan zou variabele *X* die nog niet is gedefinieerd tot deze fout leiden:

```

4 X 3
SYNTAX ERROR
4 X 3
^

```

Welke fout zou men krijgen indien *X* wel zou zijn gespecificeerd als $X \leftarrow 5$?

Vooraf bij het werken met vectoren is er nog een andere fout mogelijk. Zouden we bijvoorbeeld bij de berekening van de kosten per prijsklasse met behulp van twee vectoren, vergeten hebben dat de vector *PRIJS* in plaats van vijf nu zeven elementen heeft, dan kregen we het volgende resultaat:

```

.6 .6 .6 .75 × PRIJS
LENGTH ERROR
0.6 0.6 0.6 0.75 ×PRIJS
^

```

De twee vectoren moeten van dezelfde grootte zijn, willen we deze berekening goed laten verlopen.

Een nog grotere fout doet zich voor bij het gebruik van een getal of berekening waarbij de functie en de variabele die een numeriek of alfanumeriek gegeven zijn kan, niet samen kunnen worden gebruikt. Het delen door nul bijvoorbeeld is niet toegelaten zodat daarvoor de volgende foutmelding ontstaat:

```

A←2
14÷(6-(4+A))
DOMAIN ERROR
14÷(6-(4+A))
^

```


Appendix A geeft een wat uitvoeriger toelichting op de foutmeldingen die kunnen voorkomen.

1.7 Oefeningen

1. Wat is het antwoord van de volgende berekeningen? (verifieer uw antwoorden via *APL*)

$3+4 \cdot 1$	$(((2 \div 3) \times 4) \div 2)$
$2 \times 6 + 7$	$200 \times 14 \div 7 - 1 + 9$
$0 \div 0$	$100. + 14,9$
$(17-5) \div 6 \times 4 - 3$	$(100+14),9$
$1 \div 2 \times 3 + 4 - 5$	$2 \times 4 \div 2 \quad 1$

2. Welke van deze namen zijn toegelaten in *APL*? (verifieer uw antwoorden via *APL*)

<i>ZERO</i>	<i>X1</i>
<i>*U5</i>	<i>X1.2</i>
<i>NAAM</i>	<i>VAR01</i>
<i>VOORΔNAAM</i>	<i>APL</i>
<i>1STE</i>	<i>APL/TAAAL</i>

3. Schrijf in *APL* de juiste notatie voor:
- zes maal twee vijfde van twintig.
 - de som van 6 en -4 .
 - de som van de twee produkten zeven maal 2,5 en de helft van negen.
 - het produkt van de twee sommen 4,2 plus 3 en 16 maal -4 .
4. a. De snelheid van het licht is ongeveer 300.000 km per seconde. Schrijf in *APL*-notatie de berekening voor de afstand die het licht in één maand aflegt, en druk dat resultaat uit in miljoenen km.
- b. Schrikkeljaren komen eens in de vier jaar voor. Schrijf in *APL* de notatie die het aantal dagen berekent tussen 1 januari 1970 en 31 december 1986, inclusief deze twee dagen. 1984 is een schrikkeljaar!
5. Gegeven zijn de volgende variabelen:

$P \leftarrow 1 \quad 2 \quad 4 \quad 8$
 $Q \leftarrow 1 \quad 2 \quad 3$
 $Z \leftarrow .5$

Geef aan welke van de berekeningen foutief zijn in *APL* en bereken het juiste resultaat. (Verifieer uw resultaat via *APL*)

$2 + P$	$P + Q$
$P \div 2$	$P \div 2 \times Z$
$Q \times Q$	$P \times 3 \times Q$
$Q \times Z$	$Q \times 1 \div Z$
$Z + P$	$P - (Z + Q)$

6. a. Schrijf de berekening in *APL* voor de conversie van graden Celsius (*C*) naar graden Farenheit (*F*). Hiervoor moet 32 bij 9/5 maal de graden Celsius worden opgeteld.
 b. Maak van *C* nu een vector met de gemiddelde maandtemperaturen. Werkt dezelfde conversie nog? Waarom?
 c. Schrijf de notatie voor de conversie van Farenheit naar Celsius.
7. Gegeven zijn de volgende variabelen:

$$\begin{aligned} P &\leftarrow 9 \quad 2 \quad 4 \quad 8 \\ Q &\leftarrow 1 \quad 2 \\ Z &\leftarrow 2 \end{aligned}$$

Geef aan welke van de berekeningen in *APL* foutief zijn en bereken het juiste resultaat. (Verifieer uw resultaat via *APL*)

P, Q	$P \times Q, Q$
Q, Z	$Z \div Q$
$Z \times P, Q$	$Z + P, Q$
Q, P, Z	$Z, P \times Q$

8. Gegeven de vectoren met alfanumerieke informatie:

$$\begin{aligned} A &\leftarrow 'IS' \\ B &\leftarrow 'HET' \\ C &\leftarrow 'JUIST' \\ D &\leftarrow 'DAT' \\ E &\leftarrow 'IK \quad ZEG' \end{aligned}$$

Welke zinnen resulteren door de samenvoeging van de vectoren:

B, A, C	E, D, B, C, A
E, B, A, C	$A, B, C, '??'$
E, C	A, D, C
$C, ' ', ' ', E, D$	$E, ' ': ' ', C$

9. Gegeven de variabelen:

```
A ← 'NAAM'
B ← 14 12
C ← 11 22 33
D ← 4
```

Welke fouten komen er voor bij de volgende berekeningen?

$B \times C$	$7 \div 3 + D$
A, B	$A, 4$
$D \times C$	$(B \times D) + D, D$
$A + A, 'VAL'$	$'MIJN', 'NAAM'$

10. Het bruto nationaal inkomen van een land is over een periode van tien jaar in miljoenen uitgedrukt:

503 602 650 750 780 900 1100 1300 1400 1450

De uitgaven voor sociale voorzieningen waren respectievelijk:

13,2 17,4 20,6 25,1 31,9 40,6 55,2 64,1 70,3 75,11

Maak van deze gegevens twee vectoren en bereken welk percentage de sociale voorzieningen van het bruto nationaal inkomen voor elk jaar vertegenwoordigen.

2 Notaties voor gegevensverwerking

In ons dagelijks leven hebben we heel veel te maken met dingen die uit de wiskunde voortkomen. Gelukkig staan we daar niet altijd bij stil want dan zouden eenvoudige berekeningen zoals delen en vermenigvuldigen, in verband met regels en overeenkomsten, mogelijkerwijze vragen doen rijzen. Toch gebruiken we allemaal dezelfde symbolen, zoals $+$ en $\times$. In *APL* is het deelteken wel anders dan een schuine streep of een horizontale lijn maar daar went men nogal snel aan, vooral als nu en dan eens een calculator wordt gebruikt. Deze notaties in *APL* zijn dus niet vreemd en werden in het eerste hoofdstuk reeds gebruikt. Een kijkje op het toetsenbord leert dat er heel wat vreemde tekens zijn. De lezer zal echter spoedig merken dat de naam van een teken heel aardig overeenkomt met het symbool zelf. Gelukkig maar want dan hoeft men bij het schrijven van programma's niet zo dikwijls naar een referentielijst te grijpen.

Tot nu toe hebben we de woorden 'teken', 'symbool', 'karakter' of zelfs '*APL*-karakter' wat door elkaar gebruikt. Dit geeft geen problemen bij het begrijpen van wat er uitgelegd wordt. In *APL* moeten we er echter om denken tekens of symbolen met een vaste naam aan te duiden. Indien de tekens of symbolen tot de taal behorende functies voorstellen die direct voor bewerkingen beschikbaar zijn, dan spreken we van *primitieve functies*. Een onderscheid wordt gemaakt tussen rekenkundige, logische, vergelijkings- en selectiefuncties. We zullen ook zien dat een bepaalde functie op twee verschillende manieren kan worden gebruikt, namelijk op een *monadische* manier dit wil zeggen opererend op een enkele variabele of getal, of op een *dyadische* manier, opererend op twee variabelen of getallen. In de monadische vorm staat de variabele of het getal altijd *rechts* van de functie. Zo'n variabele of getal noemen we verder een *gegeven* van de functie. In de dyadische vorm komt een gegeven links *en* rechts van de functie voor. Uiteraard moeten men altijd eerst goed onderscheid maken tussen variabelen die één enkel getal of alfanumeriek gegeven bevatten, en die variabelen die er meerdere bevatten. Deze laatsten kennen we reeds als vectoren.

2.1 Scalair en vectoren

Gegevens worden in *APL* meestal opgeslagen in variabelen met een beschrijvende naam. Dit hebben we gezien bij het maken van een reeks getallen voor prijzen. Zolang deze namen volgens de toegelaten syntax gebruikt worden kunnen we ze volgens een verschillende rangorde definiëren.

Eén enkele waarde heet in *APL* een *scalaire*. Deze waarde kan een getal zijn of een letter. De constante 3.14 en de letter Z zijn dus scalairen. De waarde 3.14 kan echter ook toegekend worden aan een bepaalde variabele zoals in $P \leftarrow 3.14$. De variabele *P* is dan ook een scalaire. Een getal is dus het kleinst mogelijke gegeven dat aan een variabele kan worden gegeven. Men kan de scalaire variabele dus eigenlijk zien als een cel waarin het gewenste getal is opgeslagen. In *APL* is een scalaire een vector met de rangorde nul. Dit wil zeggen dat het geen domein heeft en dus geen reeks van getallen kan bevatten. De variabele heeft geen dimensie.

Anders ligt het bij datgene wat we een vector noemen. Hierbij worden er aan variabelen verschillende getallen of alfanumerieke teksten toegewezen. We hebben gezien dat functies konden werken met scalairen en vectoren. Dit houdt echter wel in dat men bij gebruik altijd moet weten of een variabele een vector van een bepaalde rangorde of een scalaire voorstelt. Vooral bij vectoren van verschillende lengte wordt dit nog wel eens vergeten.

De dimensie van een vector is één, en de lengte of grootte van deze vector wordt bepaald door het aantal elementen of getallen in die vector. Zo is de variabele *PRIJS* een vector van zeven elementen. Er bestaan ook vectoren van dimensie (of rangorde) twee en zelfs hoger. Zoals een vector de samenvoeging is van verschillende scalairen, zo is een vector van dimensie of rangorde twee een samenvoeging van vectoren van dimensie één. De samengevoegde vectoren moeten allemaal van dezelfde lengte zijn. Zo'n vector noemt men een matrix. Aangezien in *APL* de functies op scalairen en vectoren werken zullen we er ook rekening mee moeten houden wat de dimensie is van vectoren.

De flexibiliteit van *APL* laat ons ook toe vectoren te definiëren van dimensie drie en hoger. Deze zijn een samenvoeging van een aantal vectoren van dimensie twee, mits deze van dezelfde lengte zijn. Specifiek wil dit zeggen dat de samengevoegde matrices een zelfde aantal rijen en kolommen dienen te hebben. Bij de meeste *APL*-systemen bestaat er geen limiet aan de dimensie van een vector. In dit boek, en vooral in hoofdstuk 5, zullen we niet verder gaan dan vectoren van rangorde drie.

2.2 Monadische en dyadische bewerkingen

We hebben gezien dat 'primitieve' functies tot de taal behorende functies zijn die door een enkel *APL*-symbool worden voorgesteld. Functies voeren dus bewerkingen uit op gegevens. Een onderscheid dient gemaakt te worden tussen functies en operatoren omdat deze laatsten bewerkingen uitvoeren op gegevens die zelf functies zijn. Met gegevens bedoelen we dus zowel waarden, variabelen als functies. Dit wordt duidelijk in de voorbeelden in verdere hoofdstukken. *APL*-functies kunnen werken op datgene wat er alleen rechts van staat, of op hetgeen links én rechts staat. In het eerste geval noemt men de functie een *monadische*

functie, en in het tweede geval een *dyadische* functie. Het feit dat een functie zowel monadisch als dyadisch kan zijn vergroot uiteraard de mogelijkheden van de functie en de taal. Dit is echter niet zonder consequenties. Het maakt een verschil of de ene of de andere vorm wordt gebruikt. Dit wordt duidelijk aan de hand van een paar voorbeelden. Laten we de reeds bekende functies op een monadische en dyadische wijze gebruiken en de resultaten bekijken. Voor de duidelijkheid zijn de functies of de zelfde lijn gezet:

	DYADISCH		MONADISCH
	2+7		+7
9		7	
	18-3		-3
15		-3	
	2×7		×7
14		1	
	18÷3		÷3
6		.33333333	

De resultaten van de monadische vorm behoeven hier nadere uitleg. We zien dat het plus-teken werd gebruikt alsof er $0+7$ stond, en dit is inderdaad 7 gebleven. Net zo werd het min-teken geïnterpreteerd alsof er $0-3$ stond, en werd het resultaat -3. Bij het $\times$ teken werd er eigenlijk niets vermenigvuldigd want de betekenis van het teken is nu veranderd. Het resultaat geeft namelijk aan of het getal in kwestie positief, nul of negatief is. Als antwoord daarop geeft de computer een 1, een 0 of een +1. Het deelteken is ook van betekenis veranderd van 'delen door' naar '1 gedeeld door'. Het resultaat is dan ook 1 gedeeld door 4 of .25. Het gebruik van het deelteken in de monadische vorm geeft zoals aangetoond de reciproke waarde van het rechtse gegeven.

In bovenstaand voorbeeld is heel duidelijk het verschil gebleken tussen de tekens - en $\bar{}$. Het normale minteken - wordt als functie gebruikt en het ietwat hoger gedrukte teken $\bar{}$ duidt een negatief getal aan. Men kan bijvoorbeeld van een positief getal een negatief getal aftrekken als

$$42 - \bar{16}$$

58

Wanneer functies zowel dyadisch als monadisch kunnen worden gebruikt dan wordt dat in dit boek meteen aangeduid. Hoe kunnen we echter het dubbele gebruik uit elkaar houden? Het antwoord hierop is eenvoudig: Uit de context is op te maken wat er is gebruikt. Dit gebeurt volgens een eenvoudige regel: Als een *APL*-functie voorkomt, en het teken onmiddellijk *links* daarvan is ook een functie, of er staat niets, dan is de *APL*-functie in kwestie in zijn monadische vorm, anders in zijn dyadische vorm.

Een opmerking bij het laatste voorbeeld is hier wel nodig. Het negatieve teken $-$ is geen functie. Deze regel is van toepassing op de volgende berekeningen waar uiteraard de rechts-naar-links regel geldt:

$$\begin{array}{rcl}
 & 16 & + \ 21 \div 7 \\
 19 & & \\
 & 6 & + \ 2 + \div 5 \\
 8.2 & & \\
 & 4 & + \ 2 - - \ 5 + \ 3 \\
 14 & &
 \end{array}$$

Het eerste voorbeeld is het gebruik van $\div$ in dyadische vorm, terwijl in het tweede voorbeeld de monadische vorm gebruikt wordt omdat het $+$ teken er onmiddellijk links van staat. Bij het derde voorbeeld tonen we aan dat de functie links van de functie dezelfde functie kan zijn. Hier wordt de min-functie dus eerst monadisch en dan dyadisch gebruikt. De berekening verloopt als volgt: $5 + 3$ is 8. Er moet nu beslist worden of het rechtse min-teken als dyadisch moet worden beschouwd. Aangezien echter het min-teken onmiddellijk links daarvan ook een functie is, wordt de monadische vorm toegepast. Dit betekent uiteraard dat het resultaat rechts negatief wordt, dus -8 . Het linkse min-teken is dyadisch te gebruiken omdat links ervan een getal staat. Het negatieve getal wordt nu afgetrokken van 2 wat leidt tot 10. Het getal 4 daarbij opgeteld geeft dan het gegeven antwoord 14.

2.3 Eenvoudige primitieve functies

We gebruiken *APL* voorlopig nog als calculator en de functies die we bestuderen zijn niet samengesteld, dit wil zeggen we gebruiken ze zoals ze op het toetsenbord voorkomen.

2.3.1 Machtsverheffing

Voor het vermenigvuldigen hebben we het teken $\times$ gebruikt, terwijl in veel andere programmeertalen de asterisk of het teken $*$ wordt gebruikt. In *APL* is dit teken gereserveerd voor machtsverheffing in de dyadische vorm. In de monadische vorm wordt het gebruikt voor een exponentiële macht. In het eerste geval is het getal links van $*$ het getal dat verheven moet worden, en het rechtse getal (of variabele) de macht. In het tweede geval wordt het linkse getal verondersteld de wiskundige konstante e te zijn (2.71828...) dat tot de macht gespecificeert door het rechtse getal verheven wordt. In de praktijk gebeurt dit als volgt:

$$\begin{array}{rcl}
 & 2 & * \ 3 \\
 8 & & \\
 & * 2 & \\
 7.389056099 & &
 \end{array}$$

Men kan dus voor heel wat verassingen komen te staan als in de dyadische vorm de $*$ door de $\times$ vervangen wordt!

2.3.2 Afronding, minimum en maximum

In enkele programma's zal het voorkomen dat we na een hele berekening als eindresultaat een decimaal getal krijgen. Men kan nu besluiten dit decimale getal naar een geheel getal af te ronden. Voor het afronden naar beneden gebruiken we in de monadische vorm het teken $\lfloor$. Het horizontaal streepje aan, de langere vertikale lijn staat onderaan in tegenstelling tot het teken $\lceil$ waar het bovenaan staat. In dat geval rondt men naar boven af. Ook in de dyadische vorm worden beide tekens gebruikt. Indien links en rechts een variabele of een getal staat, dan betekent het teken $\lfloor$ dat het minimum van de beide getallen als resultaat is gewenst. Het gebruik van het teken $\lceil$ zou echter het maximum aangeven. Dit is hieronder geïllustreerd:

	$A \leftarrow 5.2$		
	$B \leftarrow 9$		
	$C \leftarrow 6.7$		
	$D \leftarrow A, B$		
	<i>DYADISCH</i>		<i>MONADISCH</i>
	$A \lfloor B$		$\lfloor A$
5.2	$A \lceil B$	5	$\lceil A$
9	$C \lfloor D$	6	$\lfloor D$
6 9		5 9	

Alle voorbeelden zullen voor zich spreken, alleen het laatste behoeft enige toelichting. De functies kunnen ook op vectoren worden toegepast, en niet alleen op scalair. De dyadische vorm met de vectoren vergelijkt hier de getallen in C en D waar deze laatste een samenvoeging is van de getallen in A en B . Bij vectoren wordt per element met de vector vergeleken en krijgt men als resultaat dat 6 het grootste is van het eerste getal van elke vector en dat 9 het grootste is van de tweede getallen.

2.3.3 Indiceren en getallen genereren

Vectoren worden in *APL* dikwijls gebruikt en kunnen soms heel lang zijn. Als naar een bepaald getal in een vector moet worden gezocht, kan men de positie aanvragen van het betreffende getal in de vector. Dit doet men met het indexteken

of een ι . In de dyadische vorm schrijft men dan de vector links en het te indiceren getal rechts van het indexteken. In de monadische vorm, die wellicht meer gebruikt wordt, spreken we ook van indiceren, maar dan op een generatieve wijze. Men kan namelijk een hele reeks van getallen gaan genereren door het teken te laten volgen door een enkel getal. *APL* genereert dan een reeks van alle gehele getallen vanaf 1 tot het getal zelf. De vector die op deze manier ontstaat kan dan worden gebruikt om zelf als index te dienen. Met deze functie zijn heel wat variaties mogelijk zoals hieronder blijkt:

	DYADISCH	MONADISCH
	$Z \leftarrow 'ALLES'$	$\iota 0$
	$V \leftarrow 1 \ 4 \ 7 \ 14 \ 16 \ 2.3 \ 7$	
	$V \iota 7$	$\iota p 1 \ 2 \ 3$
3		1 2 3
	$V \iota 15$	$\iota 7$
8		1 2 3 4 5 6 7
	$V \iota 2.3 \ 7$	$10 + \iota 5$
6 2		11 12 13 14 15
	$Z \iota 'L'$	$\iota 'A'$
2		DOMAIN ERROR

In de monadische vorm moet het rechtse getal een scalaire of een scalaire variabele zijn. Ook het getal nul is toegestaan. Dit geeft aanleiding tot het creëren van een lege vector. Zou men echter deze vector als $L \leftarrow 0$ definiëren, dan zou bij samenvoeging later van andere getallen of vectoren, deze 0 altijd als eerste element aanwezig zijn. Dit is niet het geval als de vector gedefinieerd is als zijnde leeg.

In elk geval geldt de regel dat alleen een scalaire mag worden gebruikt en dat een vector van welke dimensie ook in een fout resulteert. In de dyadische vorm is het wel toegestaan een vector van welke dimensie dan ook te gebruiken. Als het getal of de getallen niet in de vector voorkomen dan wordt als index een getal gegeven dat één groter is dan de lengte van de vector. Uit de voorbeelden is ook gebleken dat het niet altijd om getallen, maar ook om alfanumerieke gegevens gaat. Uiteraard zal bij een karaktervector een karakter rechts van het indexteken moeten staan. Verder valt hierbij nog op te merken dat bij het monadisch gebruik de karaktervector niet is toegelaten zoals de foutmelding ook aangegeven heeft, en dat het rechtse gegeven een geheel getal moet zijn. Decimalen zijn dus in de monadische vorm niet toegelaten.

2.3.4 Elementenindicering

Getallen of alfanumerieke gegevens kunnen ook door een logische functie worden bepaald. Daartoe kan men het teken $\in$ gebruiken. Dit kan zowel met scalair als

vectoren, maar alleen in dyadische vorm. Enkele voorbeelden maken dit duidelijk:

```

V ← 1 4 6 8 10
Z ← 'ABCDE'
V ∈ 4
0 1 0 0 0
V ∈ 9
0 0 0 0 0
Z ∈ 'AE'
1 0 0 0 1
'D' ∈ Z
1

```

Zoals blijkt is de vector rechts de gegeven vector waarin moet worden gezocht en vormt de scalaire of de vector links de vraag of het een element van de vector rechts is. Rechts kan per definitie echter ook een scalaire staan, alhoewel dat minder wordt toegepast.

2.3.5 Toevalsgetallen

Het vraagteken hebben we reeds gebruikt als een alfanumeriek gegeven in het voorbeeld van de opbouw van een zin met vectoren. Als functie heeft het teken ? zowel een monadische als een dyadische betekenis. Monadisch geeft het een toevalsgetal weer dat tussen 1 en het getal rechts van de functie gelegen is. Dit toevalsgetal, of *random*-getal, is altijd een geheel getal. Indien decimale random-getallen moeten worden gegenereerd, dan kan men de gehele getallen delen door een macht van 10. In de dyadische vorm heeft het teken ? de betekenis van het 'delen' zoals bij het trekken van speelkaarten. Het rechtse getal geeft de limiet aan van 1 tot dat getal, en het linkse getal bepaalt hoeveel getallen er willekeurig moeten worden uitgekozen.

DYADISCH		MONADISCH	
	4?52		?1
19 27	5 37	1	
	4?52		?100
31 44	4 43	91	
	2 ? 5		?100
4 1		14	
	5?5		? 10 10
3 4 1	5 2	7 4	

Bij beide vormen mag het rechtse getal een scalaire of een vector zijn. Zoals duidelijk blijkt hoeft het resultaat van één en dezelfde instructie niet éénduidig te zijn. Bij het gebruik van het teken ? wordt intern in de computer altijd een vector gecreëerd met het rechtse getal. Het indexteken in zijn monadische vorm wordt dus zo gebruikt dat bij 1100 eerst een vector wordt gegenereerd door 1100 en daaruit wordt een willekeurig getal gekozen. De vector is echter intern gegenereerd en wordt zodoende niet door APL weergegeven.

2.3.6 Absolute waarden en restwaarden

Soms werken we bij berekeningen met getallen alleen met hun absolute waarde, dus ongeacht of de waarde positief of negatief is. Dit kan zich bijvoorbeeld voordoen wanneer men alleen maar een bepaalde afwijking wil weten, ongeacht of die afwijking naar boven of naar beneden is. In de monadische vorm wordt daarvoor het teken van de absolute waarde gebruikt, namelijk |. Dit wordt in de onderstaande voorbeelden met scalaren en vectoren duidelijk:

	⁻ 6.95		⁻ 14 1 12 ⁻ 10
6.95		14 1 12 10	
	S+412		V+9 ⁻ 8 7 ⁻ 6
412		9 8 7 6	

In de dyadische vorm heeft de functie | een andere betekenis. Wanneer men gaat delen krijgt men vaak een decimaal getal als resultaat. Dit resultaat wordt dan vaak nog eens afgerond naar boven of beneden. Bij het programmeren komt het soms voor dat men bij bepaalde delingen niet geïnteresseerd is hoeveel keer een getal op een ander getal deelbaar is, maar juist hoeveel er nog overblijft, dus de 'rest'. Daartoe gebruiken we de functie | zoals in:

	3 9		2 4 5 6 7 8
0		0 1 0	1 0
	4 9		7 18 50 1 ⁻ 9
1		4 1 1	5
	4 10		2 10
2		0 1 0	1 0 1 0 1 0 1

Het gebruik van deze rest-functie is toe te passen bij het berekenen van wisselgeld, ervan uitgaande dat tot op de cent wordt terugbetaald. Veronderstel dat er 91 cent terug betaald moet worden. Hoeveel kwartjes, dubbeltjes, stuivers en centen zijn

dat, als we proberen eerst zoveel mogelijk kwartjes te gebruiken, dan de dubbeljes, enz.? De berekening hiervoor verloopt als volgt:

```

KWARTJES+⌊91÷25
NOGΔOVER+25|91
DUBBELTJES+⌊NOGΔOVER÷10
NOGΔOVER+10|NOGΔOVER
STUIVERS+⌊NOGOVER÷5
CENTEN+5|NOGΔOVER
KWARTJES,DUBBELTJES,STUIVERS,CENTEN
3 1 1 1

```

Hierbij is het bedrag dat nog over is telkens aangepast.

Bij negatieve getallen dient men wel op te letten. Als in de dyadische vorm het linkse gegeven van | negatief is, dan is het resultaat net zo alsof het positief was geweest. Maar als het rechtse gegeven negatief is en het linkse gegeven positief, geeft het resultaat aan hoeveel er tekort is om het linkse getal een geheel aantal maal op het rechtse getal te delen. Dit resultaat is negatief. De voorbeelden hieronder maken dit duidelijk:

-3	$-7 18$	-4	$-7 ^{-}18$
	$7 18$		$7 ^{-}18$
4		3	

2.3.7 Lengte en vorm van vectoren

Bij de vorige bespreking over vectoren hebben we gesproken over de lengte en de dimensie van deze vectoren. In programma's is het veelal nuttig te weten of een bepaalde functie niet tot een fout zal leiden door bijvoorbeeld twee vectoren van verschillende lengte met elkaar te vermenigvuldigen. Om deze lengte te bepalen, gebruiken we in de monadische vorm de functie ρ . Dit houdt in dat voor de vector die zich rechts van het teken bevindt, de lengte wordt bepaald, dit wil zeggen hoeveel elementen, getallen of alfanumerieke karakters, zich in die matrix bevinden. Het aantal prijsklassen van een menu in een restaurant kan dus bepaald worden aan de hand van deze lengte bepalende functie:

```

      ρPRIJS
7
      ρA+1

```

De lengte van een scalaire heeft een leeg resultaat, net zoals het resultaat van een lege vector is gedefinieerd als 0. Evenals van een vector de lengte kan worden aangegeven, kan ook de vorm van de vector en de dimensie worden aangegeven.

Hiervoor gebruiken we in de dyadische vorm de functie ρ . Het rechtse getal blijft een scalaire of een vector, terwijl het linkse getal de lengte van de vector aangeeft. Indien het linkse gegeven uit meerdere getallen bestaat dan geven deze de lengte van elke dimensie in een vector aan. Dit wordt duidelijk bij onderstaande toepassingen:

```

      A ← 5 ρ 3
      A
3 3 3 3 3
      □ ← B ← 7 ρ 1 10
1 2 3 4 5 6 7
      C ← 2 2 ρ 1 2 4 6 8 10 12 14 16
      C
      1 2
      4 6
      □ ← D ← 3 3 ρ 'JANEETVIS'
JAN
EET
VIS

```

Het blijkt dat het aantal getallen links van de functie de dimensie van de vector aangeeft. Een tweedimensionale vector noemen we een matrix en een links gegeven met drie getallen resulteert in een driedimensionale matrix. Als slechts één enkel getal voorkomt, dan bepaalt dit de eigenlijke lengte van de vector. In het bovenstaande voorbeeld is de vector B gegenereerd door de 1, een vector met tien elementen. Het getal links bepaalt echter dat slechts de eerste zeven getallen worden gebruikt.

Het resultaat van de lengtebepaling door het monadische gebruik van de functie kan ook op vectoren van grotere rangorde worden toegepast. Als het om een matrix gaat dan resulteert dit in twee getallen die respectievelijk de lengte van de rijen en de kolommen in de matrix aangeven. Deze twee getallen vormen dus ook een vector. Hierdoor kan de monadische vorm van ρ tweemaal worden toegepast, zodat de tweede keer de twee getallen de input vormen, wat een resultaat van twee geeft. Waarde twee is nu de dimensie of rangorde van de matrix. Op enkele van de bovenstaande voorbeelden hebben we de monadische vorm van de ρ functie als volgt toegepast:

	ρA		$\rho \rho A$
5		1	
	ρB		$\rho \rho B$
7		1	
	ρC		$\rho \rho C$
2 2		2	
	ρD		$\rho \rho D$
3 3		2	

Zodra een matrix is gevormd en er bewerkingen mee uitgevoerd zijn, is het soms passend er weer een ééndimensionale vector van te maken, bijvoorbeeld om ergens op een externe gegevensbank weg te kunnen schrijven. Er zal dus een mogelijkheid moeten bestaan om de matrix te 'ontbinden'. Dit wordt onderstaand toegelicht.

2.3.8 Samenvoegen en ontbinden

Het gebruik van de functie om scalairen en vectoren samen te voegen hebben we in het vorige hoofdstuk reeds uitvoerig besproken. Hiervoor werd in de dyadische vorm de komma gebruikt. In de monadische vorm heeft het gebruik van de komma een bijna omgekeerde betekenis. Het teken kan toegepast worden op scalairen, vectoren en matricen. Het resultaat van de bewerking is altijd een vector. Dus, het ontbinden van een scalaire geeft als resultaat een vector van één element. Het ontbinden van een vector geeft weer een vector, en het ontbinden van een matrix levert een vector op waar alle rijen in één vector naast elkaar komen te staan.

Nemen we bijvoorbeeld de matrix M en de scalaire S , dan kunnen we het ontbinden als volgt toepassen:

```

      M ← 4 3 p 1 12
      M
      1 2 3
      4 5 6
      7 8 9
      10 11 12
      ,M
1 2 3 4 5 6 7 8 9 10 11 12
      M ← 3 4 p 1 12
      ,M
1 2 3 4 5 6 7 8 9 10 11 12

```

De matrices M zijn van verschillende dimensie, en geven toch hetzelfde resultaat, namelijk die ene vector. De reden waarom scalairen in vectorvorm worden gezet is omdat bepaalde functies alleen maar op vectoren van toepassing zijn. Aangezien een vector van één element ook toegestaan is, blijkt het ontbinden een handige manier te zijn.

2.4 Logische functies

Bij het programmeren komt het vaak voor dat we willen nagaan (of testen) of bijvoorbeeld een variabele een bepaalde waarde heeft, of een vector een bepaald

getal bevat. Deze testen kunnen worden uitgevoerd met behulp van logische functies. Het resultaat van een logische test is altijd een logisch getal, ofwel 1 (test positief) of 0 (test negatief). Dit hebben we reeds gezien bij de toepassing van de functie $\in$. Na het vragen of het linkse geveent een *element* is van het rechtse getal of een reeks getallen,, werd het resultaat steeds als een 0 of 1 weergegeven.

Met uitzondering van de laatst besproken functie in dit hoofdstuk, zijn alle logische functies in de dyadische vorm te gebruiken.

2.4.1 AND en OR functies

In de logica kan men de vraag stellen of aan twee voorwaarden is voldaan, of aan één van de voorwaarden is voldaan. Hiertoe gebruiken we de functies $\wedge$ en $\vee$. Bij het gebruik van de AND functie wordt het resultaat 1 indien aan beide zijden van de functie een logische waarde 1 wordt gevonden, anders wordt het resultaat 0. Dit illustreren we aan de hand van de vier mogelijkheden:

	$0 \wedge 0$		$1 \wedge 0$
0		0	
	$0 \wedge 1$		$1 \wedge 1$
0		1	

Bij de OR functie wordt het resultaat als 1 gegeven als ten minste één van de logische waarden aan beide zijden van de functie 1 is, zoals blijkt uit:

	$0 \vee 0$		$1 \vee 0$
0		1	
	$0 \vee 1$		$1 \vee 1$
1		1	

Het gebruik van deze functies zal in bepaalde toepassingen relevanter zijn dan in andere. Ook zullen ze alleen maar mogen worden gebruikt als de te vergelijken waarden logische getallen 0 of 1 zijn. vectoren of andere scalaires zijn dus niet toegelaten.

2.4.2 De functies $=$, $\neq$, $>$, $\geq$, $<$ en $\leq$

Het gebruik van deze functies ligt voor de hand, en zij zijn op zowel scalaires als

vectoren van toepassing. De interpretatie of naam van deze functies is als volgt:

= : *GELIJK AAN*
 ≠ : *NIET GELIJK AAN*
 > : *GROTER DAN*
 < : *KLEINER DAN*
 ≥ : *GROTER OF GELIJK AAN*
 ≤ : *KLEINER OF GELIJK AAN*

De onderstaande voorbeelden spreken voor zichzelf. Wel moet aandacht worden geschonken aan het feit of men met een scalaire of een vector werkt:

0	7 = 4	1	3 ≤ 5
1 0 1	7 = 7 1 7	0 0 1	3 ≤ 1 2 3 6
1	7 ≠ 4	0	3 ≥ 5
0 1 0	7 ≠ 7 1 7	1 1 1	3 ≥ 1 2 3 6
1	7 > 4	1	3 < 5
0 1 0	7 > 7 1 7	0 0 0	3 < 1 2 3 6

2.4.3 De ontkenningfuncties $\sim$, $\wedge$ en $\vee$

De meeste logische functies leveren een 0 of 1 op, of een logische vector met waarden 0 en 1. Men kan dit resultaat ook omkeren zodat een 1 een 0 wordt, en een 0 een 1. Hiervoor gebruiken we het ontkenningsteken $\sim$ in de functies NOT, NAND en NOR. Voor de laatste twee functies typt men het teken $\sim$ boven op het teken $\wedge$ en $\vee$ door middel van de BACKSPACE toets, of door het teken $\sim$ vóór de logische berekening te zetten. De onderstaande voorbeelden geven aan wat hiervan het resultaat is:

1	~ 0	0 1 0	~ 1 0 1
1	0 $\vee$ 0	0	1 $\vee$ 0
0	0 $\vee$ 1	0	1 $\vee$ 1
1	0 $\wedge$ 0	1	1 $\wedge$ 0
1	0 $\wedge$ 1	0	1 $\wedge$ 1

Zoals reeds eerder opgemerkt is de functie $\sim$ alleen in zijn monadische vorm te gebruiken. Met de kennis van deze logische functies zal het nu mogelijk zijn eenvoudige *APL*-programma's te schrijven. Om echter het materiaal van dit hoofdstuk goed onder de knie te krijgen zijn de onderstaande oefeningen van groot belang.

2.5 Oefeningen

1. Bereken met de hand het resultaat van elk van de volgende bewerkingen en verifieer uw antwoord daarna via *APL*:

$$\begin{aligned} 4 \times 7 \div 2 \\ 6 - - 4 \div \div 4 \\ 18 \div 3 - - 2 \\ 1 \times \times 6 \end{aligned}$$

$$\begin{aligned} 2 * 3 \\ 6 + 1.5 + \div 2 \\ 18 \div 3 - - 2 \\ 4 + \times - 1 \end{aligned}$$

2. Gegeven de volgende scalair en vectoren:

$$\begin{aligned} A &\leftarrow 6.5 \\ B &\leftarrow A, A, 4.5 \\ C &\leftarrow 2 \ 2 \rho 4 \ 2 \ 1 \ 0 \\ D &\leftarrow 2 \ 2 \rho 1 \ 0 \ 1 \ 0 \\ E &\leftarrow 'ABC' \end{aligned}$$

Wat is het resultaat van de bewerkingen:

$$\begin{aligned} 1 \ \lfloor \ A \\ \lceil B \\ 3 \ 1 \rho E \\ C + D \\ C - D \\ A \ \lfloor B \\ 1 \ 3 \rho E \\ E \ 1 \ 'D' \\ E \in 'D' \end{aligned}$$

$$\begin{aligned} (1 \ 0), \ \rho B \\ 10 - 2 + 1 \ 3 \\ \rho \rho C + D \\ C \times D \\ C \div D + 1 \\ \rho, C \\ 3 \ 2 \rho (, D), , C \\ 'D' \ 1 E \\ 'D' \in E \end{aligned}$$

3. Interpreteer de volgende instructies in *APL* en voer ze uit op een terminal:
 - a. Bepaal vijf willekeurige getallen tussen 1 en 10.
 - b. Bepaal vijf willekeurige getallen tussen 0 en 1.
 - c. Bepaal een willekeurig getal tussen -5 en 5.
 - d. Hoe weet men of de prijsklasse 9.95 op de menukaart voorkomt?
 - e. Maak een matrix waarvan de eerste rij de prijsklassen en de tweede rij de winst per prijsklasse bevat.

- f. Hoeveel keer is 13942 deelbaar door 433 en wat is de rest?
 - g. Bepaal hoeveel elementen er zijn in een matrix waarvan de vorm $\begin{pmatrix} 6 & 3 \end{pmatrix}$ is?
4. Genereer met één instructie de volgende matrices:
 - a. Twee rijen en twee kolommen met vier getallen tussen 100 en 200.
 - b. Drie rijen en een aantal kolommen zodat de woorden EEN, TWEE en DRIE in elkaar opvolgende rijen komen te staan.
 - c. Vier rijen en vier kolommen waar alle elementen 0 zijn, uitgezonderd de waarden op de diagonale as, die 1 zijn (dit is een identiteitsmatrix.)
 - d. Rangorde 3, elke lengte is 2 met de getallen 2,4,8,16,... die worden gegenereerd door middel van de index-functie $\uparrow$.
 - e. Vier rijen en één kolom met het totaal aantal elementen in de matrices bepaald in a,b,c en d.
 5. Welke instructies zijn nodig om te bereken hoeveel briefjes van 25, 10 en 5 gulden, en ook het aantal gehele guldens in muntstukken dienen te worden terugbetaald bij een aankoop van negen gulden indien men met 100 gulden betaalt?
 6. Bereken eerst met de hand het resultaat van de volgende bewerkingen en controleer uw antwoord daarna via *APL*.

$$0 \vee 1$$

$$0 = 0$$

$$1 = 0$$

$$1 \wedge 1$$

$$0 \leq 1$$

$$1 \geq .5$$

$$7 < 1$$

$$7 > 1$$

$$0 \vee 1 = 0$$

$$0 = 0 \geq 0$$

$$1 \neq 0 + 1$$

$$1 \wedge \sim 1$$

$$0 \leq 1 > 3 < 4$$

$$1 \geq .5 \times 2 + 1 > 2$$

$$0 \vee 1 \wedge 1$$

$$\sim 0 \vee 1 \neq 0 < 2 \times 4 > 6$$

7. Gegeven de scalairen:

$$A \leftarrow 2$$

$$B \leftarrow 4$$

$$C \leftarrow 8$$

Schrijf met één enkele instructie de test in *APL* neer.

- a. Zijn A en B groter dan 0?
- b. Is A maal B groter of gelijk aan C?
- c. Is B kleiner dan 6 en groter dan 3?
- d. Is A gelijk aan C gedeeld door B, of is C groter dan A en B samen?
- e. Is A groter dan 1, is B groter dan A, en is C groter dan B?

8. Gegeven de vectoren:

$$\begin{aligned} D &\leftarrow \begin{pmatrix} 4 \\ 6 \end{pmatrix} - \begin{pmatrix} 2 \\ 4 \end{pmatrix} \\ E &\leftarrow \begin{pmatrix} 2 \\ 3 \end{pmatrix} \\ F &\leftarrow \text{'ZOEK'} \\ G &\leftarrow \text{'ZEER'} \end{aligned}$$

Wat zijn de logische vectoren als resultaat van:

- a. Is D kleiner dan E?
 - b. Is D kleiner of gelijk aan E?
 - c. Is F gelijk aan G?
 - d. Is E groter dan D?
 - e. Is D groter of gelijk aan E?
 - f. Is F niet gelijk aan G?
9. a. Veronderstel dat bij bestellingen groter dan 500 gulden, een bedrijf aan een klant twee procent korting geeft. Bereken in één instructie het bedrag dat de klant moet betalen. Gebruik hiervoor een variabele voor het bedrag en een test om te zien of deze variabele groter is dan 500.
- b. Schrijf een instructie die berekent hoeveel pakken papier men nodig heeft om 950 boekjes te drukken van elk 21 vellen, gegeven dat een pak papier 500 vel bevat.
10. Schrijf twee instructies voor het oplossen van een kwadraatsvergelijking waarvan de algemene vorm $aX + bX + C = 0$ is, zodat de waarde(n) van X wordt (worden) gevonden.

3 Gedefinieerde functies en controle

In de inleiding hebben we gesproken over programmeren in *APL*. De programma's die in de *APL*-taal worden geschreven, noemt men *functies*. Dit lijkt een beetje verwarrend met wat we in hoofdstuk 2 primitieve functies hebben genoemd. Het onderscheid zit in de oorsprong van een functie. Primitieve functies zijn reeds gedefinieerd in *APL*, met andere woorden de *APL*-gebruiker hoeft deze niet meer aan de hand van een programma te schrijven. Neem het voorbeeld van het genereren van getallen zoals in 1.10. Door het samenvoegen van de getallen 1 tot en met 10 zouden we hetzelfde resultaat krijgen, zij het dan met meer invoer. Deze primitieve functies zijn op zichzelf heel krachtig en we zullen ze dan ook zoveel mogelijk benutten.

APL-programma's echter worden *gedefinieerde* functies genoemd omdat ze door de programmeur door middel van een bepaalde naam worden gedefinieerd. Zo'n functie is een verzameling van uitdrukkingen in *APL* die de gegevens volgens de gewenste wijze verwerkt. Zo kunnen we in het voorbeeld van het wisselgeld, al deze instructies in een gedefinieerde functie genaamd *WISSEL* stoppen, zodat we bij ieder gebruik van die naam het wisselschema van 91 cent krijgen. We zullen dus voortaan de *APL*-programma's *functies* noemen, dit in overeenstemming met de bedoeling van de taal. Het onderscheid tussen gedefinieerde functies en primitieve functies zal blijken uit de context.

Ook een functie met de naam *WISSEL* lijkt niet zo interessant als men er alleen maar het wisselschema voor 91 cent mee kan uitrekenen. Als we het aantal centen variabel, aan de functie zouden kunnen toevoegen, bijvoorbeeld in monadische vorm zoals we dat bij de functies ook doen, en we zouden het bedrag in een variabele stoppen, dan kunnen we in feite de functie algemeen gebruiken. Om dit laatste mogelijk te maken zullen we in dit hoofdstuk ingaan op het gebruiken van functies in verschillende vorm, zoals monadisch en dyadisch gebruik, het gebruik van gedefinieerde functies binnen een andere gedefinieerde functie, de invoer en uitvoer binnen een functie en het onderscheid van lokale en globale variabelen.

3.1 Het definiëren van een functie in *APL*

Het begin van de definitie van een functie begint met een daarvoor in *APL* gereserveerd teken, namelijk de omgekeerde driehoek ∇ boven de letter *G*. Zodra

dit teken is ingevoerd, gaat het werk in *APL* niet meer als een calculator functioneren, maar als geprogrammeerde functie. Zo kunnen we bijvoorbeeld de naam van de functie als *WISSEL* definiëren, of een andere functie als *VERKOOP*, zoals het analyseren van de verkoop in een restaurant:

```

∇WISSEL
∇VERKOOP

```

Als na invoer van bijvoorbeeld de naam *WISSEL* de *RETURN*-toets gebruikt wordt, neemt *APL* aan dat de functie gedefinieerd is. Dit is de eenvoudigste manier om een *APL*-functie te definiëren. Zodra dit gebeurt, komt *APL* terug met regelnummer 1, zoals in:

```

          ∇WISSEL
[1]

```

Het regelnummer is altijd tussen vierkante haakjes gegeven en is altijd nummer 1 als het de eerste keer is dat deze functie gedefinieerd wordt.

Wanneer het regelnummer er staat kan men een *APL*-instructie invoeren zoals we dat op de calculatorische wijze reeds deden. We kunnen bijvoorbeeld als eerste regel een variabele *GELD* specificeren met de waarde van 91 cent. Dat gebeurt dan als volgt:

```

          ∇WISSEL
[1]      GELD←91
[2]

```

Onmiddellijk na het indrukken van de *RETURN*-toets komt *APL* met het volgende regelnummer en weer kan een instructie worden gegeven. Dit gaat zo door totdat we alle gewenste instructies hebben ingevoerd:

```

          ∇ WISSEL
[1]      GELD←91
[2]      KW←⌊GELD÷25
[3]      X←25⌊GELD
[4]      D←⌊X÷10
[5]      X←10⌊X
[6]      ST←⌊X÷5
[7]      C←5⌊X
[8]      KW,D,ST,C

```

Op regelnummer 9 moeten geen instructies meer worden ingevoerd, men kan de functie afsluiten door opnieuw het teken ∇ in te voeren:

[9] ∇

Dit afsluiten van een gedefinieerde functie kon eigenlijk ook al op regel 8:

[8] *KW, D, ST, C* ∇

Het gebruik van het teken ∇ opent en sluit dus een functie.

De functie *WISSEL* is nu gedefinieerd in het werkgeheugen en kan uitgevoerd worden. Dit doen we heel eenvoudig door het intypen van de naam van de functie. We roepen als het ware de gedefinieerde functie op met de door ons gekozen naam. De *APL*-vertaler treedt onmiddellijk in werking en voert alle instructies in de functie uit. Het enige dat we te zien krijgen is het antwoord als resultaat van de uitvoering van regel [8]:

WISSEL
3 1 1 1

Als tweede voorbeeld maken we een eenvoudige analyse van de verkoop van een aantal schotels in een restaurant. We weten reeds dat er zeven verschillende prijsklassen zijn. De werkelijke kosten zijn in percentages uitgedrukt. Aan het eind van een dag heeft men ook berekend hoeveel schotels van elke prijsklasse verkocht zijn. Aan de hand van deze gegevens wil men berekenen wat de totale omzet is, wat het totaal aantal geserveerde schotels, wat de gemiddelde omzet per schotel, wat de gemiddelde winst per schotel en wat de totale winst van die avond is.

We hebben nu al de mogelijkheden om dit uit te rekenen als we per prijsklasse de prijs geven, het aantal geserveerde schotels per prijsklasse en de percentuele kosten per prijsklasse. We voeren deze in zoals we al eerder deden, dus niet in een gedefinieerde functie maar gewoon in het werkgeheugen:

<i>PRIJS</i> ←	14.95	19.95	22.95	26.95	29.95	34.95	64.95
<i>SCHOTELS</i> ←	9	12	14	17	14	6	2
<i>KOSTEN</i> ←	.6	.6	.7	.7	.6	.6	.6

We noemen de functie *VERKOOP* en berekenen al het gevraagde in de volgorde zoals hierboven. We zullen echter gebruik maken van een operator die we tot nog toe niet hebben gezien. Deze operator heet de reductie-operator $/$. Deze wordt in het volgende hoofdstuk nader verklaard. Voor onze berekeningen hier gebruiken we de reductie samen met de functie $+$. Als deze beide tekens $+$ en $/$, in die volgorde vóór een vector worden geplaatst, dan betekent dit dat we een *som reductie* maken van deze vector. Dit komt neer op het maken van een som van al de

elementen in de vector. Deze som is een scalaire hetgeen betekent dat de vector is gereduceerd tot een getal. Als we bijvoorbeeld de som willen berekenen van het aantal geserveerde schotels op een dag, dan kan dit met:

+ / SCHOTELS
74

Bij het maken van de APL-functie zullen we aannemen dat de winstmarge hetzelfde is als 1 min de percentuele kosten. Om de uitvoer wat aantrekkelijker te presenteren zullen we vectoren met daarin de bijbehorende tekst geven zodat die tijdens de uitvoer ook op het scherm of op het papier verschijnen. De functie wordt dan op deze manier ingebracht:

```

▽ VERKOOP
[1]  'TOTALE OMZET'
[2]  + / PRIJS × SCHOTELS
[3]  'TOTAAL GESERVEERDE SCHOTELS'
[4]  + / SCHOTELS
[5]  'GEMIDDELDE OMZET PER SCHOTEL'
[6]  (+ / PRIJS × SCHOTELS) ÷ + / SCHOTELS
[7]  'GEMIDDELDE WINST PER SCHOTEL'
[8]  (+ / PRIJS × SCHOTELS × 1 - KOSTEN) ÷ + / SCHOTELS
[9]  'TOTALE WINST'
[10] + / PRIJS × SCHOTELS × 1 - KOSTEN
▽
    
```

Bij de uitvoering van de functie komt elke ingevoerde regel overeen met een uitvoerregel, en wel als volgt:

```

                VERKOOP
TOTALE OMZET
1912.3
TOTAAL GESERVEERDE SCHOTELS
74
GEMIDDELDE OMZET PER SCHOTEL
25.84189189
GEMIDDELDE WINST PER SCHOTEL
9.283445946
TOTALE WINST
686.975
    
```

Ditzelfde programma kan elke dag gebruikt worden, en zo lang de prijsklassen niet veranderen of de kosten per prijsklasse, dan moet alleen de vector met het aantal schotels per prijsklasse worden ingevoerd. Deze drie vectoren worden door

de functie gebruikt alhoewel ze niet in de functie zijn gespecificeerd. Dit komt omdat in *APL* de functie zelf opzoekt wat de variabelen zijn die nodig blijken te zijn voor bepaalde berekeningen. De drie variabelen worden ook globale variabelen genoemd omdat ze door de functie in het werkgeheugen, het vaste geheugen, worden opgeroepen. In verdere stappen van het programmeren zullen we zien dat er een verschil bestaat tussen globale en lokale variabelen.

3.2 Typen van gedefinieerde functies

Gedefinieerde functies kunnen, net als bij primitieve functies, ook monadisch en dyadisch zijn, afhankelijk van de gegevens links en rechts van de functienaam. Bij de voorgaande voorbeelden van de door de programmeur zelf gedefinieerde functies werden noch rechts noch links gegevens geplaatst. In die gevallen noemt men de functie *niladisch*. De functies *VERKOOP* en *WISSEL* zijn voorbeelden van niladische functies.

We kunnen dit type functie ook definiëren met een expliciet resultaat zodat bij het gebruik ervan een variabele, of bij gebrek daaraan de uitvoer, het berekende resultaat meekrijgt. Zo kunnen we bijvoorbeeld de functie *WISSEL* als een functie definiëren die als resultaat het aantal overgebleven centen geeft. Bij gebruik in andere functies zou men bijvoorbeeld rekening kunnen houden met het feit dat centen bijna niet meer uitbetaald worden en dat men volgens een bepaalde afspraak afrondt. De definitie van de functie zou er dan als volgt moeten uitzien:

$\nabla C \leftarrow WISSEL$

Telkens als de functie gebruikt wordt is het resultaat, namelijk het aantal centen, onmiddellijk te gebruiken door een andere functie die dit getal nodig zou kunnen hebben.

Functies kunnen zoals reeds gezegd ook monadisch zijn. Zoals bij primitieve functies heeft de functie alleen een rechts gegeven. Dit gegeven kan een getal, karakter, of een vector zijn, en is aangeduid met een variabelenaam. Bij de functie die het wisselgeld berekent kan men het bedrag in munten dat moet worden teruggegeven, ook reeds op de regel van de definitie van de functie invoeren. Dit doen we als:

$\nabla WISSEL\ GELD$

De eerste regel in de functie is dan eigenlijk niet meer nodig zodat we de functie kunnen gebruiken door het intypen van de functienaam gevolgd door een spatie en het bedrag. De berekening verloopt dan verder normaal omdat de variabele *GELD* nu meteen als invoervariabele werkt. Ook hier kunnen we gebruik maken van het verwijzen naar een expliciet resultaat.

In de dyadische vorm worden twee gegevens aan de functie toegekend en deze moeten bij de definitie van de functie gedefinieerd worden. Als we bijvoorbeeld in het voorgaand voorbeeld ook nog het aantal gehele guldens zouden willen toevoegen, dan kan men dit vóór de functienaam geven als:

∇ GULDEN WISSEL GELD

Uiteraard is de variabele *GULDEN* niet in de functie gebruikt. Op het eind van dit hoofdstuk zullen we echter deze functie zo uitbreiden dat ook de teruggave in briefjes van 100, 25, 10 en 5 kan worden bepaald. Ook in de dyadische vorm kan men één van de resultaten expliciet aan een variabele toewijzen. Voor de drie typen van functies hebben we dus de volgende mogelijkheden MET of ZONDER expliciet resultaat:

TYPE	ZONDER	MET
NILADISCH	∇ FUNCTIONIE	∇ UIT+FUNCTIONIE
MONADISCH	∇ FUNCTIONIE X	∇ UIT+FUNCTIONIE X
DYADISCH	∇ Y FUNCTIONIE X	∇ UIT+Y FUNCTIONIE X

Samengevat kunnen we over de functie-typen zeggen dat deze monadisch of dyadisch zijn en gegevens meekrijgen op het moment dat de functie wordt opgeroepen. Functies met een expliciet resultaat kunnen binnen of buiten een gedefinieerde functie in een *APL*-instructie worden gebruikt.

Het schrijven van functies, of programmeren, zal vooral van belang zijn als bepaalde primitieve *APL*-functies door een mogelijke beperking van een micro-computer toch niet kunnen worden gebruikt. Als dit het geval is kan men een functie definiëren die exact dezelfde uitwerking heeft.

3.3 Eenvoudige invoer en uitvoer

De eenvoudigste vorm van uitvoer hebben we reeds gezien bij het op één regel schrijven van een vector met tekst, met als resultaat een berekening. We kunnen deze uitvoer elders laten weergeven dan op de regel waar de berekening plaats vindt door aan een variabele het resultaat toe te kennen. We kunnen dan of de variabele een regel verder in de functie neerschrijven, zoals we dat deden met de resultaten van het aantal kwartjes en dubbeltjes, of we kunnen de variabele opvragen nadat de uitvoering van de functie is beëindigd. Dit is mogelijk omdat

de variabelen 'globaal' zijn, ook al zijn ze al eens afgedrukt. Dit kan men als volgt bereiken:

```

                WISSEL
3  1  1  1
      KW
3
      KW × .25
0.75

```

Op de eerste regel van de functie *WISSEL* hebben we de variabele *GELD* als een aantal centen gespecificeerd. We kunnen de functie zo aanpassen, dat daar niet het aantal centen staat, maar wel een teken dat aanleiding geeft om tijdens de uitvoering van de functie te vragen hoeveel centen er terug moeten worden betaald. Dit teken is het invoerteken □, ook de *quad* genoemd. Regel [1] van de functie ziet er dan zo uit:

```
[1]  GELD←□
```

Bij de uitvoering van de functie wordt bij het van rechts naar links interpreteren het teken □ gezien met de *APL*-vertaler. Onmiddellijk wordt hierop gereageerd door eenzelfde teken gevolgd door een dubbele punt, hetzij op het beeldscherm, hetzij op het papier. Op de regel daaronder kan men dan het gewenste getal dat de centen voorstelt invoeren. De twee voorbeelden illustreren hoe dit werkt:

	WISSEL		WISSEL
□:		□:	
	19		58
0 1 1 4		2 0 1 3	
	GELD		
19			

Bij het rechtse voorbeeld hebben we nagegaan hoeveel het ingevoerde bedrag was. De variabele *GELD* is een globale variabele.

Niet alleen getallen kunnen worden ingevoerd, maar ook alfanumerieke gegevens. Hiervoor gebruiken we hetzelfde teken □ met daar boven op met behulp van de BACKSPACE-toets het teken ', zodat we het samengestelde teken □ verkrijgen. Om het gebruik hiervan te tonen, schrijven we een nieuwe functie. Deze functie laten we een zinnenetje invoeren, en nagaan hoeveel keren de letters *A* en *Z* in de

tekst voorkomen. We programmeren een functie genaamd *AZ* en voeren deze dan als volgt uit:

```

      ▽  AZ
[1]  TEKST←□
[2]  A←+/TEKST∈'A'
[3]  Z←+/TEKST∈'Z'
[4]  'AANTAL KEREN A EN Z:'
[5]  A,Z
      ▽

```

```

      AZ
DE MAN ZAG DE ZON IN DE ZEE ZAKKEN
AANTAL KEREN A EN Z:
3 4

```

Zodra het invoer teken □ wordt bereikt wacht de functie op invoer zonder dat er uitvoer wordt gegenereerd. De tekens □ en : verschijnen dus niet en alfanumerieke gegevens kunnen vanaf de meest linkse positie van de invoereenheid worden getypt. Verder is de berekening van het aantal letters als volgt: Eerst wordt in logische vorm weergegeven waar de tekstvector een bepaalde letter heeft (1) of niet heeft (0), en dan geeft de som van deze vector meteen aan hoeveel keer 1, dus hoeveel keer de letter voorgekomen is.

Invoer en uitvoer van gegevens kan ook in een berekening plaatsvinden. Als we bijvoorbeeld geen variabelenaam willen gebruiken kunnen we in plaats daarvan het invoerteken gebruiken. Bij het lezen zal de *APL*-vertaler de berekeningen waarmee hij bezig is onderbreken en om de gegevens vragen, zoals in:

```

      A←4
      2×Z←A+1B←□
□:
      6
10 12 14 16 18 20

```

Dit bovenstaande voorbeeld kan zowel in een gedefinieerde functie worden gebruikt als in een instructie met de terminal als calculator. Dit laatste gebeurt vooral wanneer buiten de gedefinieerde functie om een vector wordt gespecificeerd die dermate lang is dat deze niet op een enkele regel kan. Aan het eind van de regel wordt dan achter het laatste getal □ getypt. Als voorbeeld nemen we een matrix met zes rijen en zeven kolommen die we als volgt invoeren:

```

M←6 7p1 2 3 4 5 6 7 7 6 5 4 3 2 1 , □
□:
8 9 10 11 12 13 14 14 13 12 11 10 9 8,□
□:
15 16 17 18 19 20 21 21 20 19 18 17 16 15

      M
      1  2   3   4   5   6   7
      7  6   5   4   3   2   1
      8  9  10  11  12  13  14
    14 13  12  11  10   9   8
    15 16  17  18  19  20  21
    21 20  19  18  17  16  15

```

Een alternatief voor deze manier van invoer zou het specificeren van een aantal vectoren zijn geweest die dan door samenvoeging de matrix kunnen vormen.

Uitvoer kan ook op dezelfde regel in de functie worden aangegeven zodat tussentijdse resultaten kunnen worden afgedrukt. Hiervoor gebruikt men het teken □ gevolgd door een pijl naar links zodat de combinatie □← een uitvoer van de resultaten van een berekening of informatie *rechts* ervan weergeeft. In de functie ∇AZ kunnen we op regel [2] en [3] het resultaat van de evaluatie van de letters A en Z in de tekst laten zien, als een logische vector van 0 en 1 vóór de som-reductie wordt toegepast:

```

[2]  A←+ /□←TEKST€'A'
[3]  Z←+ /□←TEKST€'Z'

```

```

      AZ
    ALLEMAAL ZIGZAG
    1 0 0 0 0 1 1 0 0 0 0 0 0 1 0
    0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
    AANTAL KEREN A EN Z:
    4 2

```

Net zoals bij de numerieke vectoren, kan men alfanumerieke gegevens op meerdere regels invoeren. Bijvoorbeeld:

```

      WOORD←'VOOR'
      WOORD,□
    BEELD
    VOORBEELD
      WOORD,□
    DEEL
    VOORDEEL

```


Bij alfanumerieke invoer is het niet nodig aanhalingstekens te gebruiken. Worden die toch getypt, dan komen ze in de tekst te staan.

Alle bovenstaande voorbeelden van in- en uitvoer geven aan dat er van de kant van de programmeur weinig controle is op de manier waarop de gegevens worden gespecificeerd. Om bijvoorbeeld een reeks getallen mooi in een kolom te laten afdrukken met daarnaast wat tekst hebben we extra faciliteiten nodig, namelijk het indelen van invoer en uitvoer. In hoofdstuk 8 wordt deze problematiek verder besproken.

3.4 Verandering in een gedefinieerde functie

Als een *APL*-functie eenmaal geschreven is zullen er soms veranderingen in moeten worden aangebracht. Dat kunnen veranderingen in de functie zelf zijn, maar ook in de invoer van gegevens. We bespreken hier de mogelijke veranderingen waarbij de opmerking geldt dat het mogelijk is dat bij bepaalde *APL*-systemen het misschien iets anders verloopt. Men doet er goed aan de gebruikersgids die bij die bepaalde *APL*-versie hoort, te raadplegen.

Veranderingen zijn mogelijk aan de functienaam zelf of aan de manier waarop de functie wordt gebruikt. Bij al deze verbeteringen en veranderingen zal het nodig zijn regels aan te passen, regels bij te voegen of geheel weg te laten, enz. Om deze veranderingen te kunnen uitvoeren, moet men eerst de functie openen, dat wil zeggen het teken ∇ intypen gevolgd door de naam van de functie. Daarbij hebben we een viertal mogelijkheden, zoals blijkt de hieronder te bespreken soorten veranderingen.

3.4.1 Toevoegen van regels

Om in één keer een goede functie te schrijven is het altijd goed eerst de bewerkingen op papier te schrijven of zelfs alle berekeningen afzonderlijk met *APL* als calculator te testen. Ondanks een gedegen voorbereiding zal het dikwijls blijken dat een paar zaken vergeten zijn die alsnog moeten worden ingebracht of veranderd.

Het toevoegen van regels kan op twee manieren geschieden; of men voegt een regel aan de functie aan op het einde van de functie, of we plaatsen een nieuwe regel tussen twee reeds bestaande regels. Bij beide manieren moet men eerst de functie openen door het teken ∇ te laten volgen door de functienaam en het indrukken van de RETURN-toets. Nu verschijnt niet het regelnummer [1] maar wel het volgende regelnummer aan het eind van de functie. Bij de functie *WISSEL*, met als laatste regelnummer [8], krijgen we:

∇ WISSEL

[9]

Nu hebben we de mogelijkheid om op regel [9] meteen een nieuwe instructie toe te voegen. Als voorbeeld laten we de functie berekenen hoeveel munstukken er moeten worden terugbetaald. Nadat op deze manier regel [9] gegeven is komt *APL* terug met de vraag of er op regel [10] ook nog wat moet staan. Dit blijkt niet nodig en we kunnen nu de functie afsluiten of nog een regel op een andere plaats toevoegen. Bemerken we echter dat ook een regel met het te verdelen bedrag moet worden uitgedrukt en dat deze informatie tussen regel [1] en [2] moet komen, dan vragen we die nog niet bestaande regel aan als regel [1.1], of een willekeurig getal tussen 1 en 2. Dit kunnen we doen door het verwijzen naar die regel met het intypen van het regelnummer met haakjes er omheen. Deze beide aanpassingen gebeuren als volgt:

```

      ∇WISSEL
[9]   'AANTAL MUNTSTUKKEN ' ; + / KW , D , ST , C
[10]  [1.1]
[1.1] 'TERUG TE BETALEN ' ; GELD ; ' CENTEN' ∇

```

Aan het einde van regel [1.1] hebben we de functie afgesloten met het teken ∇. Meteen hebben we ook een voorbeeld gezien van het mengen van uitvoertypes, namelijk alfanumeriek en numeriek. Dat kan altijd, zolang deze door het scheidingsteken ; apart wordt gehouden. Als we nu de gehele functie willen zien dan kunnen we dit doen door opnieuw de normale wijze de functie te openen, ditmaal gevolgd door een instructie die aanleiding geeft tot een lijst van de gehele functie. Daartoe gebruiken we het teken □ tussen vierkante haakjes:

```

      ∇WISSEL[□]∇
      ∇ WISSEL
[1]   GELD←□
[2]   'TERUG TE BETALEN ' ; GELD ; ' CENTEN'
[3]   KW←⌊ GELD÷25
[4]   X←25 | GELD
[5]   D←⌊ X÷10
[6]   X←10 | X
[7]   ST←⌊ X÷5
[8]   C←5 | X
[9]   KW , D , ST , C
[10]  'AANTAL MUNTSTUKKEN ' ; + / KW , D , ST , C
      ∇

```

Hierbij merken we een paar dingen op. Zo is regelnummer [1.1] nergens meer te vinden. Dit is nu regel [2] geworden. De vroegere regel [2] is nu regel [3] geworden, enz. De laatste regel die we zojuist hadden ingevoerd als regel [9] is nu regel [10] geworden. Omdat de functie nog open staat (er is namelijk nog geen tweede teken ∇ ingevoerd als sluiting), wordt de mogelijkheid geboden alsnog veranderingen in de functie aan te brengen. Hoeft dit echter op dit moment niet, dan sluiten we de

functie af zoals hierboven. Willen we uitvoer van de functie dan hoeven we alleen maar de naam te typen, gevolgd door de terugkeertoets:

```

WISSEL
□:
      46
TERUG TE BETALEN 46 CENTEN
1 2 0 1
AANTAL MUNTSTUKKEN 4

```

3.4.2 Weglaten van regels

In de functie die telt hoeveel keer de letters A en Z in een tekst voorkomen kunnen we bijvoorbeeld regel [4] weglaten omdat we toch weten dat het eerste getal van het resultaat over de letter A gaat en het tweede getal over de letter Z. Dit weglaten kunnen we op verschillende manieren doen zoals uit onderstaande voorbeelden blijkt:

	∇AZ		$\nabla AZ[4]$
[6]	[4]	[4]	[$\Delta 4$]
[4]	$\wedge$	[4]	∇
[5]	∇		

Een eerste manier is het openen van de functie ∇AZ met, na het verschijnen van de volgende regel, de verwijzing naar regel [4]. Eenmaal daar beland kunnen we door het indrukken van ATTN- en de RETURN-toets de regel in kwestie laten wegvallen. Eventueel kan de functie daarna afgesloten worden. Een tweede manier is directer in de zin dat reeds bij het openen van de functie de verwijzing van de te verwijderen regel wordt meegegeven. De procedure met het gebruik van de ATTN toets verloopt dan op de zelfde wijze. Ook bij het invoegen van regels kan deze directe referentie bij de opening van de functie worden gebruikt. Bij bepaalde APL-systemen is het gebruik van de ATTN-toets vervangen door het commando [Δ regelnummer] waarbij het regelnummer uiteraard verwijst naar de te verwijderen regel.

3.4.3 Verbeteren van fouten in een regel

Het foutief invoeren van gegevens of instructies komt spijtig genoeg heel wat keren voor. In APL hoeven er weliswaar al heel wat minder instructies worden ingevoerd dan bij andere programmeertalen, toch gebeurt het regelmatig dat een verkeerd teken wordt ingebracht. Ontdekt men dit op het moment dat men nog aan die regel bezig is, dan kan men net zoals de methode buiten de functie, de BACKSPACE- en ATTN-toetsen gebruiken. Als we bij het invoeren van regel

[1.2] in de geldwissel-functie een paar fouten hadden gemaakt dan zou men ze alsnog als volgt kunnen herstellen:

```

          VWISSEL[1.1]
[1.1] 'TERUH TE BE'
          ^
          G TE BETALEN ';GEL;' ENTEN'
[1.2] V

```

Bij de meeste terminals is het voldoende alleen BACKSPACE te gebruiken of een andere mogelijkheid op het toetsenbord, zodat op de zelfde regel terug kan worden gegaan tot waar de fout dient te worden hersteld. Worden de fouten pas later ontdekt dan kan men de foutieve regel opvragen door, net zoals bij het invoeren en weglaten van een regel, het regelnummer tussen vierkante haakjes in te typen. In het geval dat de hele regel moet worden veranderd, kan men volstaan met deze eenvoudige oproep. Meestal echter wil men slechts een paar cijfers of letters veranderen. In die gevallen gebruikt men het teken □ voor het volledig tonen van de functie, echter ditmaal voorafgegaan door het regelnummer en gevolgd door een nummer dat de positie aangeeft van de fout:

```

          VWISSEL[2□8]
[2] 'TERUG TE BETALEN ';GELD;' CENTEN'

```

Wanneer we bij de gevraagde regel zijn beland, wordt van de programmeur verwacht dat er aangewezen wordt waar exact de fout is. Dit kan men door met behulp van de spatietoets onder de regel voort te bewegen tot men onder de fout komt. Wil men bijvoorbeeld een cijfer, letter of teken vervangen door één of meer andere cijfers, letters of tekens, dan zet men onder de fout het schuine streepje / gevolgd door het aantal cijfers, tekens of letters dat ervoor in de plaats moet komen. Moet er niets voor in de plaats komen dan geeft men geen nummer op. Gebruikt men echter geen schuine streep en wel een nummer, dan wordt er niets in de bestaande regel weggelaten, maar worden er wel zoveel spaties tussengevoegd. Deze combinaties kunnen afzonderlijk maar ook op éénzelfde regel voorkomen zoals in:

```

[.] TEMPC+(TAEMPG+32×5/9
[.□11]
          /      /2  1 /1

```


Nadat de RETURN-toets is ingedrukt wordt de te veranderen regel nogmaals weergegeven, maar nu met de nodige ruimte om er het gewenste tussen te voegen:

```
[..]      TEMPC+(TEMP  32 ×5 9
ooo
[..]      TEMPC+(TEMPF-32)×5÷9
```

(De tekens °°° hierboven dienen alleen om de regel vóór en na de invulling te scheiden.)

Als men aan het eind van de regel iets wil toevoegen dan kan men dit doen door, in plaats van een geschatte plaats van de fout aan te geven, een 0 te gebruiken:

```
          VWISSEL[5□0]
[5]  D←[X+10▽
```

We hebben hier niets aan de regel toegevoegd en meteen de functie afgesloten.

3.4.4 Veranderen van de functienaam

Bij het afsluiten van een functie worden alle regels opnieuw genummerd van 1 tot en met het volgnummer van de laatste regel. De naam van een functie kan ook via een regelnummer worden bereikt om eventuele veranderingen aan te brengen. Dit regelnummer is volgens afspraak bij alle APL-systemen het nummer 0. Men kan in deze regel, zoals we hierboven hebben gezien, alle mogelijke veranderingen aanbrengen. De programmeur kan op deze manier de functie AZ veranderen in een functienaam die iets meer beschrijvend is zoals:

```
          ∇AZ[0]
[0]  A△EN△Z ∇
```

Hierbij hebben we gebruik gemaakt van het onderlijnen van de letters A en Z. Hiermee onderscheidt deze naam zich van een gelijkwaardige functienaam zonder onderlijning. Ook de invoeging van het driehoekje △ is toegestaan omdat het geen primitieve functie is. Het wordt hier dus als een gewoon teken gebruikt.

3.5 Toetsen van condities in functies

Zoals we reeds eerder zagen is het mogelijk om een reeks van bewerkingen in een gedefinieerde functie te laten uitvoeren. Soms is het wenselijk om een reeks van bewerkingen alleen dan te laten uitvoeren wanneer aan een bepaalde voorwaarde

(of conditie) is voldaan en dat, als er niet aan die voorwaarde wordt voldaan, er geen of een andere reeks van bewerkingen plaatsvindt. Daartoe gebruiken we de reeds in hoofdstuk 2 besproken logische functies. Niet alleen kunnen we daarmee van de ene groep bewerkingen naar een andere overgaan, maar ook kunnen we nu bepaalde bewerkingen laten herhalen. Daar ligt immers de kracht van het programmeren, namelijk om de computer eenzelfde groep bewerkingen telkens opnieuw te laten uitvoeren. Alhoewel dat in dit hoofdstuk nog niet duidelijk zal blijken ligt de kracht van *APL* in vergelijking met andere talen in dit daadwerkelijk verkorten van herhalingen. Dit komt omdat de meeste operatoren en functies ook kunnen worden toegepast op scalair en vectoren, waaronder bij de laatste groep de matrices.

Om te bewijzen dat programmeren in *APL* heel handig en kort kan zijn, wordt wel eens een voorbeeld in verschillende programmeertalen gegeven, waarbij in één enkele regel kan worden geschreven wat in andere talen soms tientallen regels vergt. De beknoptheid van de getoonde *APL* functie is dan vooral te danken aan het feit dat er geen toetsen voor tellers in herhalende bewerkingen nodig zijn. Toch lijkt het vaak wenselijk een zekere vorm van controle te hebben over hetgeen in de functie moet worden uitgevoerd. We nemen als voorbeeld een functie die als doel heeft gegevens in een vector te laten invoeren. Dit kan buiten de functie om gebeuren, maar om verschillende redenen wil men controle uitoefenen op de correctheid van de in te voeren gegevens. Bijvoorbeeld de getallen die worden ingevoerd moeten tussen de 1 en 100 zijn, inclusief deze twee getallen, en er mag een maximum van 20 getallen mag worden ingevoerd. Verder kan men gerust twee of drie getallen in één keer invoeren, maar zodra er 20 getallen zijn ingevoerd loopt de functie ten einde. Er is echter een probleem met het invoeren van verschillende getallen. Als één van die getallen niet aan de voorwaarden voldoet, dan worden de andere ook niet aan de vector toegewezen.

Doorgaans maakt een programmeur eerst een schets van de functie, in de vorm van een diagram of een structuurschema. We zullen stapsgewijs de te volgen bewerkingen uitleggen:

1. Definieer de vector als een lege vector
2. Lees de numerieke gegevens
3. Als in de invoer geen getal groter is dan 100 en kleiner dan 1, gaan we verder met stap zes, anders gaan we door met stap vier.
4. Geef een foutmelding, want de laatste invoer was kennelijk niet juist.
5. Ga terug naar stap twee.
6. Voeg de juiste gegevens bij de vector die moet worden opgevuld.
7. Zijn er nu reeds 20 of meer getallen? Indien dit niet het geval is ga dan terug naar stap twee, ga anders door met stap acht.
8. Geef een melding dat er nu genoeg getallen zijn.
9. Controleer of er inderdaad genoeg getallen zijn.

Om maar meteen te laten zien hoe zo'n programma in *APL* eruit ziet, volgt hieronder de gedefinieerde functie. Aan de hand van deze functie zullen de verdere toelichtingen beter worden begrepen:

```

      ▽ Z←DATA L
[1]   Z←10
[2]   A←,□
[3]   →(0=+/(A>100)∨A≤0)/6
[4]   'GETAL(LFN) NIET TUSSEN 0 EN 101 - OPNIEUW'
[5]   →2
[6]   Z←Z,A
[7]   →(L>ρZ)/2
[8]   'ER ZIJN GENOEG GETALLEN'
[9]   Z←LρZ
      ▽

```

Laten we eerst even de manier bekijken waarop we de functie, die we *DATA* noemen, hebben gedefinieerd. De letter *Z* links van de functienaam impliceert dat er een variabele zal worden gekozen voor het resultaat van de bewerking. Willen we bijvoorbeeld de vector de variabele naam *IN* geven, dan kunnen we dit bij het gebruik van de functie als zodanig opgeven in plaats van de variabele naam *Z*, die hier eigenlijk maar dienst doet als een 'nepvariabele'. Om de functie wat algemener te maken hebben we de functie monadisch gemaakt door er rechts van de naam een variabele of een getal aan toe te voegen. In de functie zelf heet deze variabele *L*, en deze duidt de lengte van de te maken vector aan. Als de functie goed werkt dan kunnen we de functie alvast oproepen door in te typen:

IN←DATA 10

We zullen nu aan de hand van de eerder vermelde stappen één tot en met negen, en de daarmee in de functie overeenkomende instructies, de resultaten van de geselecteerde regels verklaren.

De eerste stap is het definiëren van de lege vector voor het resultaat. In de tweede stap vragen we aan de gebruiker van de functie numerieke data. Dit merkt de gebruiker door het verschijnen van de tekens $\square$ en $\therefore$. De manier waarop we dit vragen is belangrijk. We veronderstellen dat meerdere getallen kunnen worden ingevoerd, en in het geval dat er maar één getal wordt gegeven, maken we van de variabele *A* een vector door vóór het invoerteken een komma te plaatsen

In de derde stap voeren we een test uit om te zien of de ingevoerde getallen aan de voorwaarden voldoen. Laten we eerst de uitdrukking tussen het pijltje naar rechts ($\rightarrow$) en de schuine streep ($/$) bekijken. Van rechts naar links lezende zien we dat $A \leq 0$ de vraag stelt of een getal kleiner is dan 1 (wat overeenkomt met het kleiner of gelijk zijn aan nul). Deze logische vergelijking levert een vector op met een


```

      □←DATA 20
□:
    10 20 30 40
□:
    14 1050 20
GETAL(LEN) NIET TUSSEN 0 EN 101 - OPNIEUW
□:
    14 10 50 20 19
□:
    22 78 99 15 23
□:
    22 24 25 29 30 78 66 45 23
ER ZIJN GENOEG GETALLEN

```

Als we nu vragen wat de vector *IN* is, dan zien we dat er 20 getallen zijn, en dat alle getallen beantwoorden aan de criteria, maar ook dat het laatst ingevoerde getal 78 niet in de vector is wat in overeenstemming is met de aangegeven lengte:

```

      IN
10 20 30 40 14 10 50 20 19 22 78 99 15
      23 22 24 25 29 30 78
      ρIN
20

```

De twee manieren van 'verspringen' in een programma zijn niet op een unieke wijze weergegeven, alhoewel de meeste *APL*-programmeurs deze wel gebruiken. Variaties die gebruik maken van logische functies zullen als onderdelen van andere voorbeelden in de verdere hoofdstukken worden getoond.

3.6 Lokale en globale variabelen

Het onderscheid is reeds enkele keren gemaakt en we zullen er in dit gedeelte alle aandacht aan besteden, vooral omdat dit onderscheid in de meeste programmeertalen niet op deze wijze wordt gebruikt.

In *APL* is een variabele een *globale* variabele als zij altijd naar dezelfde gegevens verwijst of zij nu wordt gebruikt in een functie, in verschillende functies, of ergens wordt gespecificeerd in het werkgeheugen. Een globale variabele heeft dus als eigenschap dat zij overal in het werkgeheugen kan worden gebruikt. Dit is heel belangrijk als er verschillende functies zijn die met dezelfde data moeten werken. Belangrijk is dat wanneer in een bepaalde functie een globale variabele wordt veranderd, bijvoorbeeld als zij getallen voorstelt en met een ander getal vermenigvuldigd wordt, zij in de volgende te gebruiken functies de nieuwe getallen voorstelt.

Dit laatste kan echter ernstige gevolgen hebben indien dit niet de bedoeling is. Daarom ook kunnen bepaalde variabelen 'lokaal' per functie worden gedefinieerd. De variabele is echter alleen maar *lokaal* als zij gelijktijdig met de functienaam gedefinieerd wordt. Elke keer dat de functie wordt gebruikt, gaat het geheugen van de computer een afzonderlijke plaats reserveren om de gegevens betreffende de lokale variabelen tijdelijk te behandelen. De gegevens die via deze lokale variabelen waren opgeslagen zijn dan ook niet meer te achterhalen zodra de functie beëindigd is. Omdat de variabelen tijdelijk worden gebruikt is het toegelaten dat er lokale en globale variabelen met dezelfde naam voorkomen zoals we in een paar voorbeelden zullen zien. We raden de beginnende APL-programmeur echter aan een afzonderlijke naam voor elke variabele te gebruiken, dit om verwarring te voorkomen.

We wijzigen de functie *WISSEL* om lokale variabelen te definiëren:

```

                ∇WISSEL0[0]
[0]  WISSEL GELD
ooo
                ;KW,D,T',C∇

```

Lokale variabelen worden dus op de regel van de definitie van de functie aangegeven en worden met een puntkomma van elkaar gescheiden, maar ook van de functienaam of van het rechtse gegeven.

Het uitvoeren van een functie met lokale variabelen verandert de waarden van gelijknamige globale variabelen niet. Lokale variabelen zijn immers niet meer beschikbaar na de uitvoer van een functie. Het volgende scenario moet dan ook duidelijk aangeven wat we bedoelen met lokale en globale variabelen. Het gebruik van het commando *ERASE* laat toe variabelen en dus de gegevens die daaraan zijn gekoppeld, in het werkgeheugen te verwijderen:

```

                KW,D,ST,C
1 2 0 1
                ∇WISSEL 14
TERUG TE BETALEN 14 CENTEN
0 1 0 4
AANTAL MUNTSTUKKEN 5
                KW,D,ST,C
1 2 0 1
                )ERASE KW D ST C
                ∇WISSEL 48
TERUG TE BETALEN 48 CENTEN
1 2 0 3
AANTAL MUNTSTUKKEN 6
                KW
VALUE ERROR
                KW
                ^

```


De lokale variabelen bestonden bij de eerste uitvoering van de functie ook nog als globale variabelen. Daarbij werden deze laatste niet gewijzigd door het programma. De verwijdering ervan in het werkgeheugen heeft tot gevolg dat na de tweede uitvoering de afzonderlijke waarde van de variabelen niet meer kan worden opgevraagd. Gebeurt dit toch dan resulteert dat in de aangegeven fout.

3.7 Gedefinieerde functies in functies

Het is vanzelfsprekend dat in een *APL*-functie er op de meeste regels gebruik wordt gemaakt van operatoren of primitieve functies. Dat dit gebruik niet gelimiteerd is tot de bestaande *APL*-functies maar dat zelfs gedefinieerde functies mogen worden gebruikt, vormt het onderwerp van deze paragraaf.

Om de nodige functies toe te lichten is het belangrijk twee aspecten bij het programmeren te bestuderen die enigszins verband met elkaar houden. Het eerste aspect is het indiceren van een vector. Dit wordt in het volgende hoofdstuk meer algemeen besproken. Het tweede aspect is dat van het geven van een naam aan een regel in een functie.

Indicering wordt gebruikt als men een bepaald element uit een reeks getallen wil weten. Bij de reeks prijsklassen van schotels kunnen we de prijs opvragen van de vierde schotel door in te typen:

```
PRIJS
26.95
```

De variabelenaam wordt gevolgd door tussen vierkante haakjes de gewenste positie in de vector. Het getal vier kunnen we echter ook vervangen door een variabele zoals in het onderstaande voorbeeld:

```
I←4
PRIJS[I]
26.95
```

In het laatste geval kunnen we bijvoorbeeld gebruik maken van een verandering van de variabele *I* om zo verschillende getallen van de vector door te lopen. Als *I* van één tot en met zeven varieert dan kunnen we alle getallen in de vector apart gebruiken. Geeft de index echter een getal aan dat groter is dan de lengte van de vector, of een negatief getal, dan krijgt men een foutmelding:

```
PRIJS[5]
29.95
PRIJS[14]
14.95 19.95 22.95 26.95
PRIJS[1 7]
14.95 64.95
PRIJS[8]
INDEX ERROR
PRIJS[8]
^
```

In functies waar de gekozen indexvariabele, zoals hier *I*, buiten de vector zou verwijzen, antwoordt *APL* met de bekende fout. Er zal dus een controle moeten plaatsvinden zodat de indexvariabele binnen de vastgestelde lengte een waarde heeft. Als de lengte van de geïndiceerde vector zelf door samenvoeging of onderdrukking varieert dan kan men vaststellen of de index nog goed is door de waarde van de variabele te vergelijken met de dimensie van de vector. Indien de waarde de lengte overschrijdt kan men desgewenst naar een andere regel in de functie gaan.

Deze vorm van controle hebben we reeds in de vorige voorbeelden gezien. Daarbij werd naar een bepaalde regel gesprongen zoals in $\rightarrow 2$. Na een toets of op een directe manier kan men verspringen naar een bepaald regelnummer of naar een bepaalde regelnaam. Aan de regel in kwestie kan een naam worden gegeven. Deze naam moet aan dezelfde eisen voldoen als iedere variabelenaam in *APL*. Tijdens de uitvoering van een functie krijgt de regelnaam het nummer van de regel als waarde. Deze regelnaam wordt automatisch beschouwd als een lokale variabele zodat na de beëindiging van de functie de namen in het werkgeheugen onbekend blijven. Als voorbeeld van het gebruik van een regelnaam nemen we een functie met een toets om te zien of de variabele *I* groter is dan een bepaalde waarde. Indien dit het geval is, moet naar het einde van de functie worden gesprongen:

```
[4]  $\rightarrow (I > 10) / \text{EINDE}$ 
[.]
[9] EINDE: 'DIT IS HET EINDE'
```

Hierbij heeft regel negen een naam *EINDE* gekregen die vanuit regel twee kan worden bereikt als variabele *I* groter is dan 10.

Een voordeel van het geven van een naam is vooral merkbaar daar waar functies moeten worden veranderd. Als er nieuwe regels moeten worden toegevoegd of weggelaten, dan klopt over het algemeen de referentie naar een regelnummer niet meer. Bij het gebruik van een naam echter, komt de naam vanzelf bij het nieuwe regelnummer.

Bij de verschillende definities van het programma *WISSEL* hebben we opgemerkt dat het wenselijk zou zijn deze functie als een dyadische functie te gebruiken omdat het dan mogelijk wordt het aantal guldens ook te verdelen in briefjes. Als we het programma in zijn huidige vorm nog eens bekijken, dan zien we dat er heel wat herhaling in zit.


```

      ▽ WISSEL  GELD;KW;D;ST;C
[1]  'TERUG TE BETALEN ' ;GELD; ' CENTEN '
[2]  KW←⌊GELD÷25
[3]  X←25⌊GELD
[4]  D←⌊X÷10
[5]  X←10⌊X
[6]  ST←⌊X÷5
[7]  C←5⌊X
[8]  KW,D,ST,C
[9]  'AANTAL MUNTSTUKKEN ' ;+/KW,D,ST,C
      ▽

```

Deze herhaling zou nog grotere vormen aannemen indien we dezelfde procedures zouden moeten herhalen voor de bedragen in hele guldens. Om dit wat op te vangen zullen we gebruik maken van wat men in het programmeren een *subroutine* of een *subfunctie* noemt. Eigenlijk is deze functie niet verschillend van een andere functie. Bij deze functie gaan we ervan uit dat er ergens een vector is gedefinieerd die het aantal mogelijke muntstukken bevat, en dat het te verdelen bedrag bekend is. Deze functie zouden we willen oproepen als:

```
91 VERDEELΔIN 25 10 5 1
```

Als antwoord zou dan een verdeling moeten komen van de te gebruiken muntstukken. Deze functie moeten we in een dyadische vorm schrijven en zij moet ook de verdeling aankunnen van grotere bedragen, want de procedure is tenslotte hetzelfde. De functie ziet er dan als volgt uit:

```

      ▽ Y VERDEELΔIN X;AANTAL;NOG
[1]  I←0
[2]  NOG←Y
[3]  L1:I←I+1
[4]  →(I>ρX)/FINDE
[5]  AANTAL←⌊NOG÷X[I]
[6]  NOG←X[I]⌊NOG
[7]  AANTAL;' KEER ' ;X[I]
[8]  →L1
[9]  FINDE:'-----'
      ▽

```

De variabele *I* wordt gebruikt als indexvariabele in de vector *X* die de waarde van de muntstukken (of briefjes en hele guldens) bevat, met die afspraak dat de waarden zijn gerangschikt van groot naar klein. De variabele *Y* bevat het te verdelen bedrag dat we op regelnummer [2] beschouwen als een bedrag dat nog resteert. Tussen regelnummers [3] en [8] gaan we nu berekenen wat het aantal stuks van een bepaalde waarde is, hoeveel er dan nog overblijft en we laten ook het resultaat afdrukken.

Regel [3] vergroot de indexvariabele *I* steeds met de waarde één en op de volgende regel wordt getoetst of de lengte van de vector niet wordt overschreden. Zolang deze niet wordt overschreden kunnen we doorgaan. We berekenen daarop het aantal keren dat een muntstuk of briefje dient te worden teruggegeven. In de variabele *NOG* houden we bij hoeveel nog resteert. Hierna kunnen we eventueel nog testen of het resterende bedrag nul is, omdat we dan de functie kunnen beëindigen. We laten deze stap echter als oefening. Op regel [7] drukken we het berekende aantal af, waarna we op de volgende regel terug naar de regelnaam *L1* gaan. Bij het bereiken van het einde van de functie drukken we een lijntje af.

We programmeren nu een functie die gebruik zal maken van de gedefinieerde functie hierboven. Deze nieuwe functie hoeft alleen nog de gegevens te verzamelen zoals die aan de subfunctie worden doorgegeven, en ook moeten nog enkele testen worden uitgevoerd:

```

      ▽  GULDEN ΔENΔ CENTEN;MUNTEN;BRIEFJES
[1]    BRIEFJES←100,25,10,5,1
[2]    MUNTEN← 25 10 ,5,1
[3]    →(GULDEN=0)/6
[4]    'VERDELING BRIEFJES EN LOSSE GULDENS'
[5]    GULDEN VERDEELΔIN BRIEFJES
[6]    →(CENTEN=0)/0
[7]    'VERDELING KLEINE MUNTSTUKKEN'
[8]    CENTEN VERDEELΔIN MUNTEN
      ▽

```

De subfunctie kan hier dus tweemaal worden gebruikt. Zodra het bedrag in guldens 0 is of het te betalen bedrag in kleinere muntstukken 0 is, wordt de verdeling niet uitgevoerd. Let hierbij vooral op regel zes die bij positieve uitslag van de test verwijst naar regelnummer 0. In *APL* is dat een sprong naar het eind van de functie. Een paar voorbeelden van uitvoer van deze functie volgen hieronder:

```

      124 ΔENΔ 33
VERDELING BRIEFJES EN LOSSE GULDENS
1 KEER 100
0 KEER 25
2 KEER 10
0 KEER 5
4 KEER 1
-----
VERDELING KLEINE MUNTSTUKKEN
1 KEER 25
0 KEER 10
1 KEER 5
3 KEER 1
-----

```



```

      0 ΔENΔ 91
VERDELING KLEINE MUNTSTUKKEN
3 KEER 25
1 KEER 10
1 KEER 5
1 KEER 1
-----
      0 ΔENΔ 0

```

Het gebruik van globale en lokale variabelen wordt nog eens getoond door het vragen naar de inhoud van enkele variabelen na dat de functies doorlopen zijn:

```

      MUNTEN
VALUE ERROR
      MUNTEN
      ^
      I
5
      L1
VALUE ERROR
      L1
      ^
      CENTEN
VALUE ERROR
      CENTEN
      ^

```

Het oproepen van de ene functie door de andere levert dus geen problemen op, althans niet in het voorgaande voorbeeld. Het gebruik van lokale en globale variabelen moet hier echter met een zekere discipline gebeuren. Een 'recursief' gebruik van een functie binnen diezelfde functie bespreken we hier niet. De oorsprong en toepassing zijn over het algemeen wiskundig van aard. Voor diegenen die zo'n functie willen gebruiken verwijzen we naar de oefeningen.

3.8 Oefeningen

1. Schrijf een functie die het volgende berekent en met het nodige commentaar afdruckt:
 - a. De gemiddelde prijs van een schotel in een restaurant.
 - b. Het aantal schotels die meer dan 15 gulden en minder dan 30 gulden kosten.
 - c. De som van de prijzen van de schotels die meer dan 20 gulden kosten.

2. Schrijf een functie met een expliciet resultaat die toelaat tekst in te voeren. Deze tekst mag niet langer zijn dan 100 karakters en moet ook afgedrukt worden bij de beëindiging van de functie.
3. Ontwerp en schrijf een gebruikersvriendelijke functie die zowel Fahrenheit naar Celcius of omgekeerd kan omzetten, gegeven dat de gebruiker van de functie de temperatuur invoert.
4. Schrijf, gebruik makende van de functie *VERDEELΔIN*, een functie die een gegeven aantal totale seconden omzet in uren, minuten en resterende seconden.
5. Ontwerp en schrijf de volgende functies:
 - a. Bereken het oppervlak van een driehoek. gegeven de basis en de hoogte,
 - b. Bereken de omtrek van een cirkel , gegeven de straal.
6. Herschrijf de functie *AZ* zodat het een dyadische functie wordt waar het rechtse gegeven een karakterscalaire is, en het linkse gegeven de tekst. De functie zoekt in de tekst het aantal keren dat het karakter voorkomt.
7. Veronderstel dat een gedefinieerde functie *ABC* van 20 regels gegeven is in een actief werkgeheugen. Wat zijn de instructies die moeten worden gegeven om:
 - a. De volledige functie te tonen.
 - b. De functie volledig te tonen en meteen te sluiten.
 - c. De tiende regel te tonen.
 - d. De functie vanaf de vijftiende regel te tonen.
 - e. Tussen regels negen en tien een instructie te voegen.
 - f. Een verandering op regel drie te maken.
 - g. Regel 10 te verwijderen.
 - h. De functie te sluiten zodat deze niet meer kan worden geopend.
8. Schrijf een functie met de naam *GWE* die berekent hoeveel de jaarlijkse werkelijke kosten bij uzelf zijn van het gebruik van gas, water en elektriciteit. De vector *Q* is een globale variabele met het verbruikte aantal liters water, kubieke meters gas en kWh electriciteit. De vector *R* is globaal en bevat de prijs per liter, per kubieke meter en per kWh, respectievelijk 34 cent, 82 cent en 14,1 cent. De lokale variabele *V* bevat het vastrecht per jaar: 48 gulden, 52 gulden en 64,20 gulden. Om de totale kosten te kunnen berekenen, moet u per energiebron, de verschuldigde BTW bijrekenen: 18, 4 en 18 procent. Veronderstel dat u 2200 gulden betaald hebt, hoeveel krijgt u terug of moet u bijbetalen.
9. Gegeven de volgende functies:


```

      ▽ P1
[1]   I←10
[2]   J←20
[3]   K←30
[4]   P2 I
      ▽

```

```

      ▽ P2 I;K
[1]   K←5
[2]   I←K
[3]   K P3 J
      ▽

```

```

      ▽ X P3 Y;I;K
[1]   I←X
[2]   J←10
[3]   K←Y
      ▽

```

Geef de waarde van I , J , K na:

- Regel drie van $P1$.
 - Regel twee van $P2$.
 - Regel drie van $P3$.
 - Uitvoer $P2$ 10.
 - Uitvoer 30 $P3$ 20.
10. Schrijf een niladische functie *REKEN* met een expliciet resultaat waar als invoer twee getallen gevraagd worden die groter zijn dan nul en kleiner dan elf, en gebruik dan de functies indien mogelijk $+$, $-$, $\times$, $\div$, $\lfloor$, $\lceil$, $?$, en $*$.

4 Manipulatie van vectoren

Na het bestuderen van functies die op een monadische of dyadische manier voor bewerkingen op scalaires en vectoren kunnen worden gebruikt, besteden we nu aandacht aan enkele functies die speciaal geschikt zijn voor bewerkingen op vectoren alleen. Deze functies zijn opnieuw primitieve functies en onderdeel van de *APL*-taal.

Als een vector gespecificeerd of gegenereerd is, kan men de informatie die de vector bevat gaan gebruiken. Soms staat deze informatie niet in de juiste vorm. Soms is er te veel aan informatie en is slechts een deel van de vector relevant. Bij al de mogelijke manipulaties in dit hoofdstuk, hebben we het alleen over vectoren van één dimensie, dus geen matrices van rangorde twee of hoger. De meeste functies zijn ook van toepassing op vectoren van een hogere rangorde en zij zullen op een zelfde manier in het volgende hoofdstuk worden besproken.

Enkele mogelijkheden die we hieronder nader toelichten zijn: het indiceren van vectoren, het splitsen van vectoren door middel van het weglaten of selecteren van delen van een vector, het roteren (volledig of gedeeltelijk omdraaien) en transponeren van vectoren, het sorteren, coderen en decoderen alsook de verkleinen (reduceren) en uitbreiden van vectoren. Het gebruik van deze primitieve functies zal de toepassing in *APL* nog gemakkelijker maken als we deze functies goed leren gebruiken.

4.1 Indiceren van vectoren

De mogelijkheid om een vector voorgesteld als een reeks letters of een reeks getallen te genereren is in zijn elementaire vorm reeds beschreven. Gebruik makend van de *index*functie *⌈* hebben we gezien dat een reeks getallen aan een variabele kan worden toegewezen. Onder gewone omstandigheden kan men de vector op dezelfde manier indiceren om bijvoorbeeld de eerste vier getallen op te roepen:

$$\begin{array}{l} V \leftarrow 10 \\ V[1 4] \\ 1 \ 2 \ 3 \ 4 \end{array}$$

Dit werkt altijd zolang men niet buiten de lengte van de vector zelf indiceert en zolang het indiceren begint met index 1. Men kan namelijk de beginwaarde van

het indiceren veranderen door gebruik van het systeemcommando *)ORIGIN*, gevolgd door een 0 of een 1. Zonder toewijzing van de gebruiker is de 'oorsprong' 1, dit wil zeggen het gebruik van *i* zal dan altijd beginnen met 1 en oplopen tot het gespecificeerde nummer. Is deze oorsprong echter gewijzigd in 0, dan kan er het volgende gebeuren:

```

      V ← 1 2 4 8 16 32
      A ← 3
      V[1A]
1 2 4
)ORIGIN 0
WAS 1
      1A
0 1 2
      V[1A]
1 2 4
      V[6]
INDEX ERROR
      V[6]
      ^

```

Als index mag dus ook een variabelenaam worden gebruikt en deze mag een scalaire of een vector zijn.

Het komt nog wel eens voor dat er een paar vectoren van dezelfde lengte zijn zoals bijvoorbeeld de vectoren *V* en *K*, en we in één van de vectoren een waarde moeten opzoeken. Als die waarde zich in vector *V* bevindt, dan geeft de toepassing van de indexfunctie in de dyadische vorm de positie weer van die waarde in de vector zoals blijkt uit het voorbeeld:

```

      V ← 60 64 69 70 59
      K ← 'ABCDEF'
      V 170
4

```

De waarde 70 bevindt zich inderdaad op de vierde plaats in de vector. Dit resultaat kan nu als index dienen om de corresponderende letter te vinden in de vector *K*. Dit doen we als volgt:

```

      K[V 170]
D

```

Men moet echter wel opletten dat dit niet resulteert in een index die buiten de lengte van de vector gaat. Als we bijvoorbeeld willen weten of de getallen 70 en 71 in de vector V voorkomen dan resulteert dit in een positie aanwijzing vier voor het getal 70, en zes voor het getal 71. Dit is het resultaat van de positiebepaling als het gevraagde element niet in een vector voorkomt. In die gevallen is de waarde van de index gelijk aan $1(\text{vector}) + 1$. Het vorige voorbeeld wat uitgebreider weergegeven resulteert dan in een indexfout:

```

I ← V 1 70 71
I
4 6
K[I]
INDEX ERROR
K[I]
^

```

In een gedefinieerde functie zal men daarom wellicht een controle moeten inbouwen opdat de index deze indexfout niet zal opleveren want anders stopt de functie met het uitvoeren van instructies.

4.2 Splitsen van vectoren

Om bepaalde delen uit een vector te halen of om een vector als een deel van zichzelf te definiëren, kan men gebruik maken van de twee functies $\uparrow$ en $\downarrow$, die respectievelijk elementen uit een vector kunnen selecteren en weglaten. Daarbij moeten we meteen de opmerking maken dat bij indicering willekeurige elementen kunnen worden gekozen, maar dat dit bij deze functies niet het geval is daar ze slechts van toepassing zijn op de uiteinden van de vector.

Beide functies worden altijd dyadisch gebruikt waarbij het rechtse gegeven de vector zelf is en het linkse gegeven het aantal te selecteren of weg te laten elementen aangeeft. Om elementen van een vector te selecteren gebruiken we het pijltje dat omhoog wijst zoals in het volgende voorbeeld:

```

V ← 7 2 3 6 8 9 5 4 3 1
4 ↑ V
4 0 0 0
-4 ↑ V
5 4 3 1

```

Zoals blijkt worden bij een positief links gegeven de eerste elementen geselecteerd en worden bij een negatief links gegeven de laatste elementen geselecteerd. Bij een karaktervector gebeurt hetzelfde:


```

K←'DE KAT SPRONG NAAR DE MUIS'
T←13
T↑K
DE KAT SPRONG
  7↑K
DE MUIS

```

De functie kan ook gebruikt worden om meer elementen te selecteren dan er in de vector staan. Bij een numerieke vector worden dan gewoon nullen aan de vector toegevoegd en bij een alfanumerieke vector worden er spaties toegevoegd. Dit blijkt uit de volgende voorbeelden:

```

V←14 12 7 8
6↑V
14 12 7 8 0 0
K←'MAART'
U←Z←8↑K
MAART
  ρZ
8

```

In de gedefinieerde functie *DATA* van het vorige hoofdstuk kan men, voor het inlezen van getallen tot maximaal twintig, regel negen met behulp van deze functie anders schrijven zodat:

$$Z \leftarrow L \rho Z \quad \leftrightarrow \quad Z \leftarrow L \uparrow Z$$

De functie $\downarrow$ is de tegenhanger van de bovenstaande functie omdat nu het linkse gegeven bepaalt welke elementen uit een vector moeten worden weggelaten. Indien het getal negatief is wordt het gespecificeerde aantal achteraan weggelaten. Dit blijkt uit de volgende eenvoudige voorbeelden:

```

V←14 2 7 89 17 12 20
4↓V
9 17 12 2 0
K←'ZONDAAR'
  4↓K
ZON
  3↑K
DAAR

```

4.3 Omdraaien van een vector

De volgorde van elementen in een vector is soms van groot belang bij het verwerken van gegevens. Er zijn enkele *APL*-operatoren die de mogelijkheid bieden de volgorde van de elementen in een vector te veranderen. De functie van deze operatoren wordt meestal wiskundig beschreven als rotatie. De meest voorkomende vorm is het geheel of gedeeltijk omdraaien van een vector.

Om bijvoorbeeld een vector volledig om te draaien gebruiken we een samengestelde functie, gevormd door de tekens $\circ$ en $|$ om tot de rotatie-operator $\oplus$ te komen. Het gebruik hiervan spreekt voor zich aan de hand van de onderstaande voorbeelden:

```
V←24 7 3
ϕV
3 7 4 2
K←'FRANS'
ϕK
SNARF
```

Dit is echter rotatie in de monadische vorm. Om de functie wat meer mogelijkheden te geven kunnen we gebruik maken van de dyadische vorm waar het teken $\oplus$ wordt voorafgegaan door een scalaire. Deze scalaire bepaalt het rotatiepunt. Bij het monadisch gebruik lag het rotatiepunt in de vector helemaal achteraan. De scalaire gaat nu het exacte punt bepalen zodat er geen volledige volgorde van de elementen in de vector ontstaat, maar wel een verschuiving van de elementen vóór en ná het rotatiepunt zoals in het voorbeeld:

```
V←6 1 72 4
2ϕV
7 2 4 6 1
K←'BALLAST'
3ϕK
LASTBAL
```

Net als bij het weglaten en selecteren van elementen in een vector kunnen we bij de dyadische vorm de functie laten voorafgaan door een negatieve scalaire, zodat het rotatiepunt vanaf het einde van de vector wordt geteld. Bij bovenstaande voorbeelden resulteert dit bij juiste keuze van de scalaren in dezelfde vectoren:

```
V←6 1 7 2 4
-3ϕV
7 2 4 6 1
K←'BALLAST'
-4ϕK
LASTBAL
```


4.4 Het bepalen van de volgorde in vectoren

De volgorde van elementen in een vector kan niet alleen met behulp van rotatie worden bepaald, maar kan ook met twee specifieke functies gebeuren. Alvorens daartoe over te gaan bekijken we een vector die de uitslag weergeeft van vijf studenten die een mondeling tentamen hebben afgelegd:

$$U \leftarrow 6 \ 7 \ 5 \ 8 \ 4$$

Op de vraag welke student het hoogste cijfer heeft behaald is het antwoord de vierde. Op de vraag wie het tweede hoogste cijfer behaald heeft, is het antwoord de tweede student. Zo kunnen we doorgaan tot we alle vijf studenten hebben gehad. Schrijven we de verschillende antwoorden op dan krijgen we een vector met de waarden 3 2 4 1 5. Deze vector geeft een index weer van de vector U die de *positie* van de scores van de studenten beschrijft. Dus het vierde getal is het eerste in de rij van hoog naar laag en het vijfde getal is het laatste. De operator die dit zelfde resultaat geeft is de *grade down*-functie Ψ , die samengesteld is uit de tekens ∇ en $|$. De functie kan zowel op numerieke als op alfanumerieke vectoren worden toegepast.

Het resultaat van het gebruik van deze functie is dus een vector met de index voor de getallen van de hoogste naar de laagste waarde. Als we deze indexreeks gebruiken als verwijzing naar de elementen van de vector U zelf, dan krijgen we de gewenste volgorde:

$$\begin{array}{l} I \leftarrow \Psi U \\ I \\ 4 \ 2 \ 1 \ 3 \ 5 \\ U \leftarrow U[I] \\ U \\ 8 \ 7 \ 6 \ 5 \ 4 \end{array}$$

Het effect van de laatste bewerking is dat we de vector U van hoog naar laag gesorteerd hebben. In de gedefinieerde functies gebruiken we deze manier van sorteren dikwijls, echter op een eenvoudiger manier:

$$U \leftarrow U[\Psi U]$$

De analoge functie is de *grade up*-functie $\Uparrow$ die de indicering van laag naar hoog geeft en de mogelijkheid biedt de vector van de laagste naar de hoogste waarde als volgt te sorteren:

$$\begin{array}{l} \Uparrow U \\ 5 \ 3 \ 1 \ 2 \ 4 \\ \Uparrow \leftarrow U \leftarrow U[\Uparrow U] \\ 4 \ 5 \ 6 \ 7 \ 8 \end{array}$$

In het geval men binnen de oorspronkelijke vector de posities van de in volgorde geselecteerde getallen of karakters wil bepalen, gebruikt men de functie tweemaal. Bij sorteren van laag naar hoog gebeurt dit als volgt:

$$\begin{array}{ccccccccc}
 & & & & & & & & U[\uparrow U] \\
 4 & 5 & 6 & 7 & 8 & & & & \\
 & & & & & & & & \uparrow \uparrow U \\
 3 & 4 & 2 & 5 & 1 & & & &
 \end{array}$$

Als er gesorteerd wordt dan zal het vierde getal het eerst worden selecteerd, dan het tweede, enz.

4.5 Coderen en decoderen van vectoren

Voordat we de hier relevante functies bespreken, is het goed even dieper in te gaan op de manier waarop we getallen eigenlijk voorstellen. Gewoonlijk is een getal opgebouwd uit een reeks cijfers waarbij niet alleen de cijfers van belang zijn, maar ook de positie van deze cijfers in de reeks. In ons tientallig stelsel is elk cijfer een coëfficiënt van een machtsverheffing van tien:

$$\begin{aligned}
 49 &= (4 \times 10^1) + (9 \times 10^0) \\
 743 &= (7 \times 10^2) + (4 \times 10^1) + (3 \times 10^0)
 \end{aligned}$$

Het gebruik van het tientallig stelsel is voor ons zo gewoon dat we er nooit bij stilstaan. De computer werkt intern echter met een binair of tweetallig stelsel. De bit is de kleinste vorm van informatie en kan dus twee toestanden aannemen. Als ergens in het geheugen vier bits zijn opgeslagen die in volgorde 1101 vormen, dan komt deze binaire code in het tientallig stelsel overeen met het getal:

$$\begin{aligned}
 1101 &= (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\
 &= \quad 8 \quad + \quad 4 \quad + \quad 0 \quad + \quad 1 \quad = 13
 \end{aligned}$$

Het verschil met het tientallig stelsel is dat we nu niet het getal tien als grondtal nemen, maar het getal twee.

De decoderingsfunctie $\perp$ is een dyadische functie die de conversie van de afzonderlijke cijfers naar het samengevoegde getal kan maken. Enkele voorbeelden maken dit duidelijk:

$$\begin{array}{ccccccc}
 & & & & 10 & \perp & 7 & 4 & 3 \\
 743 & & & & & & & & \\
 & & & & 2 & \perp & 1 & 1 & 0 & 1 \\
 13 & & & & & & & & \\
 & & & & 5 & \perp & 4 & 10 & 6 \\
 156 & & & & & & & &
 \end{array}$$

Aangezien een willekeurig grondtal kan worden gekozen, ligt het voor de hand dat voor toepassingen zoals het evalueren van een vergelijking waar een variabele X een bepaalde waarde heeft en waarbij er verschillende machtsverheffingen plaats vinden, de oplossing eenvoudig is zoals:

$$\begin{array}{l}
 4X^3 + 3X^2 + 7X + 2 \\
 \circ \circ \circ \text{ WORDT MET } X=2.4 \\
 2.4 \ 1 \ 4 \ 3 \ 7 \ 2 \\
 91
 \end{array}$$

De codering verloopt net omgekeerd en wordt door het gebruik van de coderingsfunctie $\top$ bereikt. Kiezen we voor een getal een ander grondtal, dan kunnen we bij het hercoderen twee verschillende technieken gebruiken, namelijk één voor gehele getallen en één voor decimale getallen.

Bij gehele getallen wordt het getal gedeeld door het grondtal waarin we het willen hebben en we noteren het quotiënt en de rest. Deze rest vormt het rechtse getal van het resultaat. Vervolgens delen we het quotiënt door het grondtal en we noteren ook hier het quotiënt en de rest. Deze rest wordt nu het tweede getal rechts van het resultaat. We gaan zo door tot het quotiënt gelijk is aan nul. We kunnen voor deze procedure een gedefinieerde functie schrijven, zoals we verderop bij de oefeningen vragen. De coderingsfunctie heeft exact deze functie en wordt ook altijd in de dyadische vorm gebruikt. Hierbij heeft het getal rechts van $\top$ altijd tien als grondtal en is het linkse gegeven een vector bestaande uit een aantal keren het nieuwe grondtal. Dit aantal wordt bepaald door het aantal delingen die op de hierboven beschreven manier dienen te worden uitgevoerd. Als voorbeeld nemen we de getallen 743 en 13 die we respectievelijk naar een getal met grondtal tien en grondtal twee coderen:

$$\begin{array}{l}
 (3p10) \top 743 \\
 7 \ 4 \ 3 \\
 (5p2) \top 13 \\
 0 \ 1 \ 1 \ 0 \ 1
 \end{array}$$

De keuze van het aantal keren het nieuwe grondtal in het linkse gegeven is van groot belang. In het laatste voorbeeld hebben we het getal in zijn binaire vorm met behulp van vijf bits laten coderen. We hadden er slechts vier voor nodig. Aangezien bij deze methode van deling de opvulling van de vector vanaf de rechter kant gebeurt, is het eerste en linkse element nul. Het decoderen van de vector met vijf elementen resulteert dan weer in het getal dertien.

Om een decimal getal van de ene basis naar de andere basis te coderen of converteren gebruiken we een andere techniek. Het gedeelte achter de komma wordt vermenigvuldigd met het grondtal waar we naar willen converteren. Dit resultaat is opnieuw een decimaal getal. We noteren daarvan het gehele gedeelte vóór de komma en vermenigvuldigen opnieuw het gedeelte na de komma met het

grondtal, enz., tot het gedeelte na de komma nul is of zo vaak als links van de dyadische functie is aangegeven. Het noteren van de gehele getallen gebeurt, in tegenstelling tot de coderingsfunctie, van de linkse tot de rechtse positie. Een voorbeeld maakt dit duidelijk. Eerst geven we een stap voor stap analyse, Dan gebruiken we in een reeks van maximaal zes getallen de functie $\top$ voor het getal .65 dat van grondtal tien naar grondtal twee moet:

DECIMAAL	$\times 2$	NOTEER
.65	1.30	1
.30	.60	0
.60	1.20	1
.20	.40	0
.40	.80	0
.80	1.60	1

$$L(6p2)\tau.65 \times 2 \times 6$$

$$1 \ 0 \ 1 \ 0 \ 0 \ 1$$

Om het getal 13.65 te coderen met grondtal twee, kunnen we eerst dertien naar een vector converteren, dan het gedeelte .65, om vervolgens de twee vectoren bij elkaar te voegen.

Bij de coderingsfunctie hoeft het linkse gegeven niet noodzakelijk een vector te zijn waarin alle getallen gelijk zijn. Dit blijkt in het voorbeeld dat we hieronder bespreken. De apparatuur in een sateliet kan ongeveer zes miljoen bits informatie op een magnetische band wegschrijven. Deze bits stellen een hele reeks foto's voor die in de ruimte genomen zijn en die naar de aarde moeten worden geseind. Deze transmissie gaat tegenwoordig met een snelheid van ongeveer 8.333333 bits per seconde. De vraag die we willen beantwoorden is: hoelang duurt zo'n transmissie? We berekenen eerst het aantal seconden dat nodig is voor de transmissie, en coderen dan volgens de verdeling 24 uur in een dag, 60 minuten in een uur, en 60 seconden in een minuut. Het resultaat wordt dan als volgt verkregen:

$$24 \ 60 \ 60 \ \tau 6E6 \div 8 \ + \ \div 3$$

$$1 \ 0 \ 0$$

4.6 Reduceren, onderdrukken en uitbreiden van vectoren

Onderdrukking van elementen in een vector wordt met behulp van de schuine streep / uitgevoerd. Het resultaat van deze functie is een verkleining van de vector. De functie is verwant met één van de weinige operatoren in de APL-taal, namelijk de reductie-operator die van de schuine streep gebruik maakt, voorafgegaan door een primitieve functie. We behandelen hier zowel de reductie-operator als de onderdrukkingsfunctie.

Als voorbeeld hadden we reeds de som-reductie van een vector gezien als het optellen van alle elementen in een numerieke vector. Dit gaven we weer door vóór de vector de tekens $+/$ te plaatsen. Alhoewel deze vorm het meest voorkomt, kunnen we verschillende primitieve functies bij de monadische vorm van de reductie gebruiken. Daarbij maken we de opmerking dat rechts van de reductie-operator een scalaire of een vector van een willekeurige dimensie mag voorkomen. In het algemeen heeft men dus een functie zoals $+$ of $\times$ die we moeten interpreteren als tussen de elementen van een vector geplaatst. Hieronder volgt een reeks voorbeelden die de mogelijkheden aangeeft:

	$V \leftarrow 102 \ 78 \ 33 \ 116$
	$A \leftarrow 50$
	$B \leftarrow 100$
	$+/V$
329	
	$-/V$
-59	
	$\div/V$
0.37202	
	$\times/V$
30455568	
	$\lceil/V$
116	
	$\lfloor/V$
33	
	$\wedge/V > B$
0	
	$\wedge/2 \mid V$
0	
	$\vee/V < A$
1	
	$>/V[2] > A, B$
1	

Hierbij enkele opmerkingen:

- De min-reductie wisselt aftrekken af met optellen, dus in feite staat er $102 - 78 + 33 - 116$.
- De deel-reductie doet hetzelfde met delen en vermenigvuldigen, dus in feite staat er $102 \div 78 \times 33 \div 116$.
- De afrondings-reductie geeft respectievelijk het hoogste getal in de vector ($\lceil$) en het laagste getal ($\lfloor$) weer.
- De operatoren $<$, $\leq$, $=$, $\geq$, $>$, $\neq$, $\vee$, $\wedge$, $\forall$, en $\exists$ kunnen voor logische vectoren worden gebruikt.
- Het laatste voorbeeld is een manier om te vragen of een gegeven getal tussen de getallen A en B valt.

Het onderdrukken van een vector lijkt wat op het opdelen van vectoren. Als linkse gegeven van de functie / gebruiken we een logische vector, waar dus alleen de getallen nul en één in voorkomen. Indien de te onderdrukken vector eenzelfde lengte heeft als de logische vector, dan wordt bij de onderdrukking elk element uit de vector weggelaten, indien daar op de respectievelijke plaats in de logische vector een nul staat. Op die manier wordt de logische vector een vector met een selecterende functie:

```

LV← 1 0 1 1 1 0 1
RV← 4 2 7 8 1 4 0
LV/RV
4 7 8 1 0
LV← 1 0 1 0 1 0 1
RK← 'ABCDEFGG'
LV/RK
ACEG

```

Onderdrukking hebben we reeds toegepast bij de controle voor springen in een gedefinieerde functie. Bij het testen op een conditie of iets aan een bepaalde voorwaarde voldeed, werd na de onderdrukkingsfunctie / een regelnummer of een regelnaam als rechtse gegeven gebruikt. In het voorbeeld hieronder vragen we of variabele A gelijk is aan één en indien dit zo is moeten we met regelnummer vier verder gaan:

```

A←2
→(A=1)/4

```

Aangezien A niet gelijk is aan één krijgen we een toepassing alsof er nu zou staan $\rightarrow 0/4$. Het resultaat daarvan is een lege entiteit en de functie zou normaal doorgaan met de bewerking op de volgende regel. Deze gedachtengang kunnen we ook volgen als we de waarde van A op verschillende mogelijkheden gaan testen, en dat, afhankelijk van het resultaat, naar het juiste regelnummer wordt gegaan. Dit illustreren we als volgt:

```

A←2
→((A=1),(A=2),(A=3))/4,14,9

```

De variabele A is gelijk aan twee en volgens de test moet de functie verder gaan met regel veertien. In feite is het resultaat van de test gebaseerd op de onderdrukking $\rightarrow 0\ 1\ 0/4,14,9$. Dit komt overeen met $\rightarrow 14$, of het gewenste resultaat.

De tegenhanger van onderdrukken is het uitbreiden van een vector. De functie die hiervoor wordt gebruikt, is het schuine streepje dat van linksboven naar rechts-onder loopt of \. Aan de hand van deze functie gebruiken we een operator die we *cumulatieve expansie* noemen. We bespreken deze operator eerst.

De som-cumulatieve expansie wordt bereikt door de tekens + en \ voor de vector te plaatsen. Als we bijvoorbeeld willen weten wat de procentuele cumulatieve omzet van een bedrijf is bij een bekende maandelijkse omzet, dan kunnen we dit stapsgewijs als volgt doen:

```

OMZET←1000×10 7.8 7.9 8.4 8.9 8.9 9.4 9 8 8 9 9.2
□←PERCENT←100×OMZET÷+/OMZET
9.6 7.5 7.6 8 8.5 8.5 9 8.6 7.7 7.7 8.6 8.8
CUMUL←+\PERCENT
CUMUL
9.6 17 25 33 41 50 59 67 75 83 91 1.0E2
    
```

Omdat we gedeeld hadden door de totale omzet, moet uiteraard aan het eind van het jaar de honderd procent bereikt zijn.

Net zoals bij de reductie-operator kunnen we nog andere vormen van cumulatieve expansie verkrijgen. Een paar voorbeelden met commentaar:

```

          -\1 2 3
1  -1 2
          >\3 2 1
3 1 1
          V←0 1 0 1 1
          v\v
0 1 1 1 1
          ≤\V
0 1 1 1 1
          <\V
0 1 0 0 0
    
```

- De min-expansie komt hier op neer dat we het resultaat van het cumulatieve aftrekken vragen. Het bereikte resultaat per element wordt dus altijd in de volgende berekening gebruikt.
- Bij logische vectoren is de \ operator meer van toepassing. In het eerste geval is het resultaat een logische vector waar vanaf de positie waar de eerste één in de originele vector voorkomt, alle verdere elementen naar rechts toe één worden. In het tweede geval worden alle elementen één zodra er een element in de originele vector voorkomt die aan de voorwaarde $\leq$ voldoet. Het derde geval beschrijft het plaatsen van alleen de eerste één die in de originele vector voorkomt, ook van links naar rechts gezien.

De uitbreidingsfunctie is analoog met het dyadisch gebruik van de onderdrukingsfunctie. In plaats van het volgens de logische vector wegnemen van elementen in een vector, worden er nu elementen aan toegevoegd. Deze elementen hebben de waarde nul in het geval dat we numerieke vectoren gebruiken, en een spatie in het geval we met alfanumerieke vectoren werken.

De logische vector moet dus groter zijn dan de vector die wordt uitgebreid. In deze logische vector komen een aantal nullen en enen voor waarbij dit laatste aantal gelijk moet zijn aan de lengte van de vector die wordt uitgebreid. Dit alles blijkt uit de onderstaande voorbeelden:

```

LV←1 0 1 0 0 1 1
EV←4 9      8 7
LV\EV
4 0 9 0 0 8 7
LV←1 1 0 1 1 1
EK←'IKOOK'
LV\EK
IK OOK

```

Het resultaat is dus weer een vector en deze vector heeft dezelfde lengte als de logische vector.

Een eenvoudige toepassing van uitbreiding is het plaatsen van een bepaald karakter tussen de letters van een woord. Door het gebruik van een 1 0 1 0 1 -vector wordt een karaktervector veranderd in een karaktervector met spaties tussen elk origineel karakter. We kunnen daarna het gewenste teken op de plaats van die spaties plaatsen. Het hele gebeuren kan men met onderstaande instructies doen:

```

TEKST←'HOERA'
NTEKST←(9p1 0)\TEKST
NSTEKST
H O F R A
NTEKST[2×14]←'!'
NTEKST
H!O!E!R!A

```

De uitbreiding kan in de dyadische vorm bij karaktervectoren het best gebruikt worden bij het weergeven van resultaten, dit om de uitvoer de gewenste esthetische vorm te geven.

4.7 Vermenigvuldigen van vectoren

In dit gedeelte worden enkele concepten besproken die, alhoewel op het eerste gezicht voor de hand liggend, soms moeilijk in *APL* te gebruiken zijn. Het gebruik van de operatoren zoals die hier zullen worden beschreven kan bij een eerste lezing overgeslagen worden, samen met het gedeelte over het vermenigvuldigen van matrices in het volgende hoofdstuk.

Er zijn twee samengestelde operatoren in *APL* die uitsluitend bij vectoren van één of meerdere dimensies worden gebruikt. Deze operatoren worden altijd in de dyadische vorm gebruikt en voeren een samengestelde bewerking op de vectoren links en rechts uit. We zien eerst die functie die men in het *APL*-jargon het *outer product* noemt, of zoals wij het kunnen vertalen, het *uitwendig produkt*. De verklaring voor deze naam volgt uit de manier waarop het resultaat verkregen wordt.

Er bestaan twee algemene manieren om vectoren te vermenigvuldigen. Men kan bijvoorbeeld element per corresponderend element vermenigvuldigen door de operator $\times$ te gebruiken. Dit kan alleen bij vectoren van *gelijke* lengte zoals we in de voorbeelden hieronder tonen:

```

      1 2 4 -8 × 2 3 4 2
2 6 16 -16
      1 2 4 -8 × 1 2 3
LENGTH ERROR
      1 2 4 -8 × 1 2 3
      ^

```

Een tweede manier is het vermenigvuldigen van ieder element uit een vector met *alle* elementen uit de andere vector. Deze vectoren mogen bij deze bewerking ongelijk van lengte zijn. De samengestelde functie bestaat uit twee *APL*-tekens en een primitieve functie. Deze twee tekens zijn: de kleinste cirkel op het toetsenbord $\circ$, en de punt $\cdot$. De operator is in dit geval aangegeven door het teken $\times$. Als voorbeeld nemen we twee ongelijke vectoren en we zien het volgende resultaat:

```

      1 2 3 ◦.× 1 2 3 4
1      2      3      4
2      4      6      8
3      6      9     12

```

in de vorm van een matrix. De dimensies van deze matrix zijn gelijk aan de dimensies van de vectoren, in de volgorde zoals ze gebruikt zijn. De rijen van deze matrix beschrijven de uitgevoerde bewerking. De eerste rij is het resultaat van de vermenigvuldiging van de eerste vector met het eerste getal in de tweede (rechtse)

vector, de tweede rij is het resultaat van dezelfde bewerking op het tweede getal van de rechtse vector, enz.

Het is echter niet nodig altijd het teken $\times$ te gebruiken. Men spreekt dan echter niet meer over vermenigvuldigen, maar over bijvoorbeeld element per element vergelijken zoals in:

$$\begin{array}{ccc} & 1 & 2 & \circ . = & 1 & 2 & 3 \\ 1 & 0 & 0 & & & & \\ 0 & 1 & 0 & & & & \end{array}$$

In principe mag men iedere primitieve functie gebruiken, al naar gelang de vereiste bewerking.

De tweede samengestelde functie heet in *APL* het *inner product* dat we hier vrij vertalen als het *inwendig produkt*. Het vermenigvuldigen van een functie gebeurt met de restrictie dat de dimensie van het resultaat kleiner is, of gelijk aan de dimensies van de bewerkte vectoren. De functie bestaat opnieuw uit een punt maar nu aan beide kanten door twee functies geflankeerd. Voor vermenigvuldiging zijn dat normaliter het teken $+$ aan de linkse kant van de punt en $\times$ aan de rechtse kant. De interpretatie hiervan is dat de elementen van de linkse scalaire of vector worden vermenigvuldigd ($\times$) met *ieder* element van de rechtse scalaire of vector en dat de resultaten van deze afzonderlijke vermenigvuldigingen worden opgeteld ($+$). Een eenvoudig voorbeeld toont dat:

$$2 \quad + . \times \quad 1 \quad 4 \quad 6$$

22

waar twee vermenigvuldigd is met één, vervolgens met vier en dan met zes. De resultaten zijn respectievelijk twee, acht en twaalf, wat gesommeerd het eindresultaat 22 geeft. Wellicht heeft de lezer wel gemerkt dat men zonder deze samengestelde functie hetzelfde resultaat had kunnen krijgen door de bewerking:

$$+ / 2 \times 1 \quad 4 \quad 6$$

22

De primitieve functies in de bovenstaande operatoren mogen ook andere zijn dan die voor samentellen en vermenigvuldigen, alhoewel deze laatste het meeste voorkomen.

4.8 Oefeningen

1. Gegeven de vectoren:

```
V ← 0 1 0 1 0
V1 ← 1 2 4 8 16
V2 ← 6 8
V3 ← 'ABCDE'
```

Wat is het resultaat van:

```
V \ V2
V / V1
V / V3
)ORIGIN 0
V1[V]
V3[V]
V3[V1[V]]
```

2. Gegeven dezelfde vectoren als in vraag 1, wat is het resultaat van:

```
)ORIGIN 1
2 + V1
- 2 + V2
3 + V \ V2
(- 2 + V3), 3 + V3
3 + - 4 + V1
0 +, + / V
```

3. Gegeven dezelfde vectoren als in vraag 1, wat is het resultaat van:

```
ΦV
ΦV1
ΦV3
- 1 ΦV3
- 2 ΦV2
- 1 ΦV / V1
```

4. Gegeven de twee vectoren:

```
S ← 6 2 4 9 1
P ← 4 8 6 1 9
```

Wat is het resultaat van:

$$\begin{aligned}
 &S+P \\
 &\Delta S \\
 &\Delta P \\
 &(\Delta S)+\Delta P \\
 &S[\Delta S] \\
 &S[\Delta P] \\
 &\Delta\Delta P \\
 &(\Delta\Delta P)+\Delta\Delta S
 \end{aligned}$$

5. Schrijf een functie waarin de volgende invoer gevraagd wordt:

- De naam van een student.
- De score van de student op een tentamen.
- De klas van de student.

Maak voor de namen een matrix en voor de scores en de klassen een afzonderlijke vector.

Laat dan de functie de nodige berekeningen maken zodat het volgende resulteert:

- Een lijst van de namen in alfabetische volgorde, gevolgd door klasnummer en score.
- Een lijst van de scores van hoog naar laag, gevolgd door de naam en de klas
- De gemiddelde score.

Gebruik als test voor deze functie de volgende gegevens:

NAMEN:	JAN	KIM	POL	BIE	ALI
SCORES:	7	9	4	8	6
KLAS:	4	4	3	2	4

- Schrijf een monadische functie *DECI* die de coderingsfunctie $\perp$ voor decimale getallen nabootst. Gebruik als test van deze functie het getal .65. Geef het resultaat weer in de vorm van een vector.
- Schrijf een monadische functie *CODE* die een getal, bijvoorbeeld 14.33 naar een waarde met grondtal twee omzet. Gebruik voor het gehele getal veertien de primitieve functie en voor het decimale gedeelte de functie *DECI* uit de vorige oefening. Geef het resultaat weer als twee op elkaar volgende vectoren, gescheiden door een punt.
- Gebruik de operator voor het inwendig produkt om het resultaat te berekenen van $A \times B$ waar:

A:	1	3	4
	5	2	6
	1	3	7

$$B: \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 0 \end{pmatrix}$$

9. Schrijf een dyadische functie met als gegevens twee matrices.

Test de mogelijkheid van het berekenen van het wendig produkt, voer de bewerking uit indien mogelijk uit en druk het resultaat af.

10. Voeg aan de functie uit oefening 9 de test toe of de ene matrix door de andere kan worden gedeeld (let op bij delen door nul!), voer de deling indien mogelijk uit en druk het resultaat af. Voeg er ook aan toe de berekening van de som van de beide matrices en bereken daarna het produkt van de sommatrix met behulp van de produkt-reductie-operator.

5 Manipulaties van matrices

De onderwerpen die in dit hoofdstuk worden besproken vormen een uitbreiding van de functies die in hoofdstuk vier werden behandeld. Een matrix kunnen we immers ook definiëren als een vector van een rangorde twee of hoger, dus met twee of meer dimensies. De voor matrices relevante functies worden hier onder dezelfde naam gebruikt als bij vectoren van één dimensie. Er zijn echter een paar functies in *APL* die alleen van toepassing zijn op een matrix. Deze zijn niet toevallig nogal wiskundig van aard. Daar waar het zonder verlies van continuïteit mogelijk is het betreffende gedeelte over te slaan zal dit worden aangegeven.

In eerste instantie moeten we de vormgeving van een matrix nog eens goed doornemen omdat dit van belang is bij de indicering van een matrix. In de meeste toepassingen komt het voor dat een matrix qua opgeslagen informatie groeit. We kunnen net zoals bij een vector, gegevens toevoegen zodat de lengte verandert. Hier zijn dus twee lengten mogelijk. Het opsplitsen van een matrix is belangrijk, niet alleen bij wiskundige toepassingen, maar ook bij die van een meer bedrijfskundige aard. Daarentegen komen de toepassingen van het roteren en transponeren van matrix niet zoveel voor.

Hetzelfde kan worden gezegd over het sorteren, reduceren of uitbreiden, delen of de inverse berekenen van een matrix. Waar mogelijk zullen voorbeelden worden gegeven om de functies volledig toe te lichten.

5.1 Definitie van een matrix

Een matrix is opgebouwd uit een aantal elementen die in rijen en kolommen gerangschikt zijn. Deze twee begrippen worden in *APL* altijd in deze volgorde gebruikt. Dit is heel belangrijk, niet alleen bij het vormen of definiëren van een matrix, maar ook bij de indicering daarvan. Met behulp van de functie ρ wordt een matrix gevormd. Het gebruik is in dit geval dyadisch en het gegeven links van de functie geeft de lengte aan van de dimensies van de matrix. De gegevens rechts vormen de elementen van de matrix:

```

M←2 3p 16
M
1 2 3
4 5 6
A←2 3p'DITZIT'
A
DIT
ZIT

```


Deze beide matrices hebben elk twee rijen en drie kolommen. Bij numerieke matrices worden de getallen uit elkaar geschreven, en bij karaktermatrices worden de karakters aan elkaar geschreven, dit alles conform het definiëren van een vector. De informatie links van de functie ρ moet een vector van minstens twee elementen zijn. Indien er zoals hierboven juist twee elementen zijn dan geeft het eerste getal aan hoeveel rijen er zijn. Het tweede getal wijst op de kolommen. Het aantal elementen in een vector is uiteraard het produkt van die twee getallen. Indien de lengte van de vector met elementen rechts van de functie ρ kleiner is dan het produkt van de twee getallen, dan worden de elementen van de vector zoveel keer opnieuw gebruikt tot alle rijen en kolommen van de matrix vol zijn. In het andere geval, waar de lengte van de vector met elementen groter is, worden alleen het aantal elementen geselecteerd dat overeenkomt met het produkt. Dit zien we als volgt:

```

      A←2 3ρ '*□'
      A
*□*
□*□

      M←2 3ρ4+110
      M
5    6    7
8    9   10

```

Het voorbeeld van de karaktermatrix wordt wel eens gebruikt bij het maken van bepaalde patronen of 'tekeningen' met de computer. In de oefeningen zijn enkele problemen van dit soort aan de orde gesteld.

Het monadisch gebruik van de functie ρ bij een matrix is net als bij een vector. Het resultaat is de lengte van elke dimensie, in de vaste volgorde van rij en kolom. Als dit resultaat wordt gebruikt om het totaal aantal elementen in de matrix te berekenen kan men gebruik maken van de produktreductie-operator en zodoende komt men tot de volgende definitie en analyse van een matrix:

```

      □←M←3 4ρ(110)*2
1.0F0    4.0E0    9.0F0    1.6E1
2.5E1    3.6E1    4.9E1    6.4E1
8.1E1    1.0E2    1.2E2    1.4E2

      ρM
3 4
      ×/ρM
12

```

Ook een lege matrix kan worden gedefiniëerd door de lengte van de dimensies op nul te stellen. Uiteraard is de variabele M wel een matrix maar zijn de dimensies nog leeg zoals hieronder blijkt:

```

      □←M←0 0p10
      ρM
0 0
      □←M←0 0p110
      ρM
0 0

```

Al deze bewerkingen zijn van toepassing op matrices van rangorde drie en hoger. Bij een drie-dimensionale matrix wordt de lengte-vector die uit drie getallen of lengtes bestaat, als volgt beschreven. Het eerste getal geeft de lengte van de eerste dimensie aan of het aantal sub-matrices van dimensie twee. Het tweede en derde getal geven voor de sub-matrix de lengte van de rij en de kolom weer. Alle sub-matrices zijn op deze manier gelijk van vorm en hebben dus een gelijk aantal elementen. Zij kunnen als een reeks matrices achter elkaar worden gezien. Als voorbeeld nemen we twee studiegroepjes die elk uit vier studenten bestaan. De namen van de studenten zijn weergegeven als vectoren met een lengte van drie karakters. Dit doen we als volgt:

```

      A←2 4 3p'JANPOLANNKIMAADJOSLEAWIL'
      A
      JAN
      POL
      ANN
      KIM

      AAD
      JOS
      LEA
      WIL

```

De twee groepjes zijn van elkaar gescheiden door een blanco regel en elk groepje bestaat uit vier studenten (rijen) met een naam van drie karakters (kolommen). Beide sub-matrices zijn dus gelijk van grootte. Als het bij het gebruik van drie-dimensionale vectoren nodig is te indiceren om bepaalde gedeelten eruit te halen, zoals een sub-matrix, dan wordt het belangrijk de exacte dimensies te weten. Hiervoor kan de functie ρ in de monadische vorm worden gebruikt.

5.2 Indicering van een matrix

Bij een vector was het voldoende één enkele index op te geven bij de selectie van een element omdat er maar één enkele dimensie bestond. Bij een matrix van rangorde twee spreekt het dus vanzelf dat er twee indexen kunnen worden gegeven, de eerste voor de rij, de andere voor de kolom. De index zelf mag gerust een vector zijn:

```

      □←M←4 4p16
      1   2   3   4
      5   6   7   8
      9  10  11  12
     13  14  15  16
      M[2;2]
6
      □←A←2 5p'ZEKERWETEN'
ZEKER
WETEN
      M[1 2;3 4 5]
KER
TEN

```

Bij het laatste voorbeeld maken we een opmerking die heel belangrijk is voor het afzonderlijk behandelen van rijen en kolommen. We hebben bij de laatste matrix *M* een rijen-index gegeven van 1 2. Aangezien er echter maar twee rijen zijn, hoeven we alleen maar te refereren naar de kolommen die van belang zijn. In dit geval hoeft er maar één index gedefiniëerd te worden (een scalaire of een vector) en *APL* veronderstelt dat deze index voor de andere dimensie geldt. Bij de karaktermatrix van hierboven wordt dit dan:

```

      □←I←2+13
      A[;I]
KER
TEN
      A[1;]
ZEKER

```

Bewerkingen van rijen of kolommen kunnen door het gebruik van een enkele index handig opgelost worden. Veronderstel dat het bijhouden van maandelijkse uitgaven in een gezin volgens een matrix op budget gebeurt. We tonen hier als oefening de bugettering van tien posten over een periode van vier maanden. We definiëren een matrix met de tien posten van (on)kosten als rijen en de vier maanden als kolommen. De getallen in de matrix zijn guldens per maand voor die post. Deze matrix die we *PRIVE* noemen, ziet er als volgt uit:

		MAAND			
		1	2	3	4
	VOEDING	960	980	1015	1010
	KLEDING	140	400	120	325
	ENERGIE	225	225	225	230
P	HYPOTHEEK	1200	1200	1200	1200
O	LENINGEN	260	260	260	260
S	VERZEKERINGEN	50	900	0	200
T	AUTO/REIZEN	220	250	1400	160
E	TELEFOON	160	0	180	0
N	HOBBY	50	100	50	100
	ONVOORZIEN	200	200	200	200

We kunnen aan de hand van deze matrix al heel wat berekeningen uitvoeren als we de juiste indicering van de matrix gebruiken. De onderstaande toepassingen worden toegelicht via het 'commentaar' karakter, voorgesteld door het teken @:

```

      @ TOTAAL VAN ALLE UITGAVEN
      +/,PRIVE
16315
      @ TOTAAL VAN MAANDEN 1 EN 3 PER POST
      +/PRIVE[;1 3]
1975 260 450 2400 520 50 1620 340 100 400
      @ VERSCHIL TUSSEN MAAND 4 EN 1 PER POST
      PRIVE[;4]-PRIVE[;1]
50 185 5 0 0 150 -60 -160 50 0
      @ VERSCHIL TUSSEN HET TOTAAL VOOR MAAND 1 EN 2
      (+/PRIVE[;2])--+/PRIVE[;1]
1050
      @ GEMIDDELDE PER MAAND AAN VASTE KOSTEN,
      @ ZOALS HYPOTHEEK, AFBETALING LENINGEN, ENERGIE,
      @ EN TELEFOON.
      .25 x +/, PRIVE[3 4 5 6 8;]
2.1E3
      @ PERCENTAGE BESTEED AAN VOEDING EN KLEDING
      @ T.O.V. DE GEMIDDELDE MAANDELIJKE UITGAVEN.
      100x(+/,PRIVE[1 2;])÷+/PRIVE
30

```

5.3 Opsplitsen van een matrix

Om elementen uit een matrix te halen hebben we hierboven gebruik gemaakt van indicering. Door het gebruik van de functie ↑ bereiken we hetzelfde effect, echter iets meer gecontroleerd en in die toepassingen bruikbaar waar men slechts een

deel van het aantal rijen en een deel van het aantal kolommen nodig heeft. Een enkelvoudig gebruik van deze functie gaat altijd via minstens één van de hoeken van de matrix, net zoals het effect bij de vectoren waar men alleen op de uiteinden kon werken. We nemen bijvoorbeeld de matrix M , die bestaat uit vier rijen en vier kolommen:

$$M \leftarrow 4 \text{ p } 16$$

$$M$$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

De functie $\uparrow$ is dyadisch gebruikt en heeft als gegeven links een vector nodig die even lang is als het aantal dimensies van de matrix. Hier hebben we een vector van twee elementen nodig. Het eerste element beschrijft de rijen die moeten worden opgenomen, het tweede element beschrijft dan de kolommen. Bij positieve getallen worden de eerste rij(en) en kolom(men) genomen, bij negatieve getallen de laatste rij(en) en kolom(men). Een toepassing op de bovenstaande matrix M levert het volgende op:

$$PV \leftarrow 3 \text{ } 2$$

$$NV \leftarrow -3 \text{ } -2$$

$$PV \uparrow M$$

1	2
5	6
9	10

$$NV \uparrow M$$

7	8
11	12
15	16

Eén van beide getallen in de vector die de keuze doet mag negatief zijn terwijl de andere positief mag zijn. Dit geeft voor de matrix A hieronder een selectie van verschillende kanten:

$$A \leftarrow 3 \text{ } 7 \text{ p 'VANDAAGVERJAARIK NIET'}$$

$$A$$

VANDAAG
VERJAAR
IK NIET

$$2 \text{ } -2 \text{ } \uparrow A$$

AG
AR

$$-2 \text{ } 3 \text{ } \uparrow A$$

VER
IK

Het gebruik van een waarde die een dimensie van de matrix overschrijdt, heeft tot gevolg dat er een rij of kolom nullen aan het opgenomen deel van de matrix wordt toegevoegd. Bij herhaald gebruik van de matrix kan men bijvoorbeeld uit de matrix M ook als volgt het 'hart' halen:

```

      -2 -2 +3 3+ M
      6   7
     10  11

```

Als voorbeeld van het gebruik van deze functie nemen we een functie die een lijst van namen opneemt. Hierbij willen we echter maar vier karakters gebruiken, ook al worden er per naam meerdere ingevoerd. Bestaat de naam echter uit minder dan vier karakters, dan worden er spaties aan toegevoegd. Op die manier kunnen we elke ingevoerde vector in een rij van de matrix stoppen:

```

      ∇ RIJ IN KOL; I
[1]  MAAND←(RIJ,KOL)ρ' '
[2]  I←1
[3]  ⑈
[4]  MAAND[I; ]←KOL↑⑈
[5]  →(RIJ≥I+I+1)/4
      ∇
      5 IN 4
JUNI
MEI
JULI
SEPTEMBER
OKTOBER

      MAAND
JUNI
MEI
JULI
SEPT
OKTO

```

De functie $\downarrow$, die wordt gebruikt voor het weglaten van gegevens in de matrix, werkt analoog aan de functie $\uparrow$ om selectief gegevens op te nemen. De matrix A als uitgangspunt nemende kunnen we met de functie $\downarrow$ de eerste rij en de eerste vier kolommen laten wegvallen zodat er een gedeelte overschiet dat nog bestaat uit twee rijen en drie kolommen:

```

      1 4 ↓A
AAR
IET

```


In het algemeen zien we dat de beide functies voor het opsplitsen van een matrix door elkaar kunnen worden vervangen, mits de gegeven vector links van de functie daarop is aangepast, of:

$$1 \ 4 \ \downarrow \ A \quad \leftrightarrow \quad \begin{matrix} -2 & -3 & \uparrow A \end{matrix}$$

5.4 Roteren en transponeren van een matrix

Deze manipulaties zijn bij matrices iets moeilijker dan bij vectoren. Zonder verlies van continuïteit kan dit gedeelte worden overgeslagen.

De functie om de rijen of kolommen in een matrix om te draaien is de rotatie-functie Φ . Bij het gebruik in de monadische vorm worden de kolommen van de matrix automatisch omgedraaid. Aangezien kolommen de tweede dimensie vormen van een twee-dimensionale matrix is het heel belangrijk te weten dat deze functie zonder verdere informatie de kolommen omdraait. Dus, de laatste kolom wordt de eerste, de eerste wordt de laatste en de kolommen er tussenin worden ook omgedraaid. Wil men echter de rijen omdraaien (ook hier verandert de dimensie van de matrix niet) dan moet men de dimensie van de rijen aangeven. Aangezien deze dimensie de eerste is voor een twee-dimensionale matrix, moet men dit als volgt doen:

```

      M←3 3p19
      M
1  2  3
4  5  6
7  8  9

      A ROTATIE RIJEN
      Φ[1] M
7  8  9
4  5  6
1  2  3
    
```

De te gebruiken dimensie wordt tussen vierkante haakjes gezet *tussen* de functie en de te bewerken matrix. Het gebruik van $\Phi[2]$ komt overeen met het automatisch omdraaien volgens de tweede (of laatste) dimensie of Φ zoals:

```

      ΦM
3  2  1
6  5  4
9  8  7
    
```

Om specifieke rijen of kolommen te verwisselen gebruiken we net zoals bij vectoren de rotatiefunctie in de dyadische vorm. Bij de matrix K hieronder kunnen we de eerste rij achteraan plaatsen door na rij één langs de eerste dimensie te roteren:

$$\begin{array}{c}
 K \leftarrow 3 \quad 4\rho \text{ 'MEERKEESWEET' } \\
 K \\
 \text{MEER} \\
 \text{KEES} \\
 \text{WEET} \\
 \\
 1\phi[1] \quad K \\
 \text{KEES} \\
 \text{WEET} \\
 \text{MEER}
 \end{array}$$

De rotatie langs de eerste kolom door middel van $\phi[2]$ of ϕ verloopt dan als volgt:

$$\begin{array}{c}
 1\phi K \\
 \text{EERM} \\
 \text{EESK} \\
 \text{EFTW}
 \end{array}$$

Soms wordt voor de rotatie over de rijen de functie $\ominus$ gebruikt. Deze functie komt overeen met $\phi[1]$. Aangezien het verschil tussen beide functies de tekens $|$ en $-$ zijn, is dit gemakkelijk te onthouden voor de rotatie langs de (horizontale) rijen en de (vertikale) kolommen. Beide functies hebben dan niet de aanduiding van een dimensie nodig.

Bij het dyadisch gebruik kan men ook het linkse gegeven als een vector definiëren. De lengte van de vector moet gelijk zijn aan de lengte van de andere dimensie langs welke men de rotatie uitvoert. In de matrix K hierboven kunnen we voor elk van de rijen (lengte vector gelijk aan $1\uparrow\rho K$) de kolommen na de gegeven rotatiepunten roteren:

$$\begin{array}{c}
 1 \quad 2 \quad ^{-1} \phi \quad K \\
 \text{EERM} \\
 \text{ESKE} \\
 \text{TWEE}
 \end{array}$$

Als we de rijen afzonderlijk voor elke kolom (lengte vector gelijk aan $^{-1}\uparrow\rho K$) volgens de gegeven rotatiepunten willen roteren, dan verkrijgen we:

1 2 1 2 $\ominus$ K
 KEET
 WEER
 MEES

Het onderwerp transpositie hebben we niet gezien bij de vectoren omdat dit begrip daar niet bekend is. Het is wel verwant aan rotatie en wordt zowel monadisch en dyadisch gebruikt. De transpositiefunctie wordt aangeduid met het teken $\ominus$, een samenstelling van de tekens $\circ$ en $\backslash$. Beginnende *APL*-programmeurs maken nogal veel fouten bij deze functie omdat ze de samenstelling per ongeluk met de tekens $\circ$ en $/$ maken. Deze functie bestaat in *APL* echter niet!

In de monadische vorm roteert de functie in één keer alle dimensies in een matrix. Voorbeeld:

$M \leftarrow 2 \ 3 \rho 16$
 M
 1 2 3
 4 5 6

 $\ominus M$
 1 4
 2 5
 3 6

 ρM
 2 3

 $\rho \ominus M$
 3 2

De dimensies zijn dus volledig omgekeerd en de elementen van de matrix zijn om een denkbeeldige as gedraaid die van links boven naar rechts onder loopt; het teken $\backslash$ geeft dus een goede indicatie van wat er gaat gebeuren.

Bij drie-dimensionale vectoren kan de verrassing echter wat groter zijn. Aangezien de dimensies volledig geroteerd zijn wordt een matrix met drie dimensies met lengte twee, drie en vier, een matrix met *drie dimensies* met lengte vier, drie en twee, zodat men van twee sub-matrices naar vier sub-matrices is overgegaan:

```

DRIE←2 3 4p'ABCDEFGHJKLMNOPQRSTUVWXYZ'
NDRIE←QDRIE
ABCD      MNOP
EFGH      QRST
IJKL      UVWX
          AM
          EQ
          IU
          BN
          FR
          JV
          CO
          GS
          KW
          DP
          HT
          LX

```

5.5 Sorteren in een matrix

Laten we voorop stellen dat er in *APL* geen aparte functie is voor het sorteren van een matrix. De reden waarom dit echter hier wel wordt besproken is om aan te tonen dat er wel degelijk in een matrix kan worden gesorteerd, om bijvoorbeeld alfanumerieke gegevens in een matrix op alfabetische volgorde te zetten, of om op de eerste rij een vector te krijgen met de kleinste getallen, enz.

Het sorteren van een vector ligt aan de basis van het sorteren van de informatie in de matrix. We kunnen dus een rij (of kolom) van een matrix beschouwen als een vector. Bij de meeste *APL*-systemen is het trouwens mogelijk numerieke en alfanumerieke gegevens op eenzelfde manier te sorteren, zoals hieronder blijkt:

```

V←14.2 16 91 44 56.4 0
V[⍒V]
0 14 16 44 56 91
A←'BCDFFA'
A[⍒A]
ABCDEF

```


Als voorbeeld voor het sorteren van een matrix tonen we de functie *SORTEER*, die voor twee-dimensionale matrices kan worden gebruikt:

```

        VSORTEER M
[1]  M
[2]  'GESORTEERD:'
[3]   $\square \leftarrow M \leftarrow M[\downarrow M, ]$ 
[4]   $\nabla$ 

```

De uitvoer van dit programma, met de afgekorte namen van de maanden, geeft:

```

        SORTEE R MAAND
MEI
JUNI
AUGU
SEPT
OKTO
GESORTEERD:
AUGU
JUNI
MEI
OKTO
SEPT

```

5.6 Reduceren, onderdrukken en uitbreiden van een matrix

Onderdrukking en uitbreiding worden net zoals bij vectoren gebruikt om de matrix te verkleinen of te vergroten. Hier komt dit dus neer op het laten wegvallen van rijen en/of kolommen of het toevoegen ervan.

We maken opnieuw gebruik van een logische vector die in lengte overeenkomt met het aantal rijen of kolommen, al naargelang de gewenste dimensie. De dimensie die men wil gebruiken wordt aangegeven na de functie: voor onderdrukking (/) of uitbreiding (\) met [1] voor de rijen, en [2] voor de kolommen. Zonder aanduiding van dimensie wordt de tweede dimensie van de kolommen verondersteld. Onderdrukking verloopt heel eenvoudig:

```

         $\square \leftarrow M \leftarrow 2 \ 3p10+16$ 
11  12  13
14  15  16
      1 0 /[1] M
11  12  13
      1 0 1 / M
11  13
14  16

```

Om in een matrix rijen tussen te voegen wordt de functie $\backslash$ gebruikt. Op dezelfde matrix M zoals hierboven kan men dit doen als:

	1	0	1	0	$\backslash M$
11	12	13			
0	0	0			
14	15	16			
0	0	0			

Bij onderdrukking en uitbreiding kan men in plaats van de eerste dimensie te specificeren als $[I]$, gebruik maken van de samenstelling van het onderdrukkingsteken $\backslash$ of het uitbreidingsteken $/$ met het minteken $-$. Dit leidt tot de volgende overeenkomst:

$V \backslash M$	$\leftrightarrow$	$V \backslash [1] M$
$V \neq M$	$\leftrightarrow$	$V / [1] M$
$V \backslash M$	$\leftrightarrow$	$V \backslash [2] M$
V / M	$\leftrightarrow$	$V / [2] M$

De reductie van een matrix verloopt zoals bij een vector en men gebruikt de reductie-operatoren zoals in hoofdstuk 4 beschreven.

5.7 Vermenigvuldigen van matrices

Alhoewel dit onderwerp voor iemand die lineaire algebra gestudeerd heeft eenvoudig lijkt, vormen bepaalde onderdelen van dit deel voor de meeste *APL*-programmeurs een moeilijkheid. Mede door het feit dat bepaalde operaties niet veel voorkomen is het soms goed een *APL* referentieboek bij de hand te hebben. De informatie hier kan gebruikt worden als referentie of als studiemateriaal dat bij een eerste lezing zonder grote consequenties mag worden overgeslagen. Aangezien het volgende deel hiermee verband houdt, kan men meteen naar hoofdstuk 6 gaan waar nog enkele primitieve functies worden besproken.

Bij het vermenigvuldigen van twee matrices moet de dimensie van de kolommen van de eerste matrix gelijk zijn aan de dimensie van de rijen van de tweede matrix. Door gebruik te maken van de operator $+. \times$ zoals bij vectoren (ook hier het *inwendig produkt*), kunnen we deze twee matrices vermenigvuldigen. Aan de hand van de operator wordt een rij van de eerste matrix vermenigvuldigd ($\times$) met de eerste kolom van de tweede matrix, waarna alle uitkomsten worden opgeteld ($+$), daarna met de tweede kolom, enz., en dat voor alle rijen. Een voorbeeld om dit te tonen: Gegeven zijn de matrices A en B , waarbij A de dimensies drie en twee heeft en B de dimensies twee en drie:

$$\begin{array}{cc} \square \leftarrow A \leftarrow 3 & 2p16 \\ 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{array}$$

$$\begin{array}{ccc} \square \leftarrow B \leftarrow 2 & 3p16 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}$$

$$\begin{array}{ccc} A + . \times B \\ 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{array}$$

De primitieve functies voor en na de punt in de operator kunnen ook hier door andere functies worden vervangen. De toepassing verloopt op dezelfde manier doordat er per rij op elke kolom een bewerking volgens de rechtse functie wordt toegepast en dat zodra deze bewerking bij een kolom gedaan is, de functie links wordt toegepast. Bij karaktervectoren is het daarom ook mogelijk vergelijkingen uit te voeren op de individuele karakters en daar dan een logische test op toe te passen. Als voorbeeld nemen we twee matrices met elk drie rijen en kolommen, waarvan de elementen uit letter bestaan. Indien we willen nagaan of er in twee matrices rijen en/of kolommen met elkaar overeenkomen, dit wil zeggen hetzelfde woord bevatten, dan kunnen we dit als volgt bepalen:

$$\begin{array}{l} \square \leftarrow A \leftarrow 3 \quad 3p 'ALSHETKAN' \\ ALS \\ HET \\ KAN \\ \square \leftarrow B \leftarrow 3 \quad 3p 'VISEETAAS' \\ VIS \\ EET \\ AAS \end{array}$$

$$\begin{array}{ccc} A \wedge . = B \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}$$

De overeenkomst bestaat hier niet en een logisch antwoord 0 resulteert voor elke vergelijking.

Bij het vermenigvuldigen van een matrix met een scalaire is het niet nodig de operator te gebruiken. Wanneer een vector wordt gebruikt kunnen we dit wel doen. De vector vormt dan in feite de enige rij en de bewerking verloopt identiek zoals uit onderstaand voorbeeld blijkt:

$$\begin{array}{cccc}
 & \leftarrow A \leftarrow 3 & 4p112 \\
 1 & 2 & 3 & 4 \\
 5 & 6 & 7 & 8 \\
 9 & 10 & 11 & 12 \\
 \\
 & 2 & 1 & 3 & +. \times A \\
 34 & 40 & 46 & 52
 \end{array}$$

Een laatste opmerking bij het gebruik van het *inwendig produkt* betreft de vorm of de dimensie(s) van het eindresultaat. De eerste dimensie van het linkse gegeven (matrix of vector) en de tweede dimensie van het rechtse gegeven bepalen de dimensie van het resultaat.

Een tweede operator voor de verwerking van twee matrices is het *uitwendig produkt* zoals bij vectoren. Net zoals daar wordt het aantal dimensies van het eindresultaat groter. We kunnen het gebruik van de functie het beste tonen door gebruik te maken van een test op gelijkheid van elementen. Men kan immers het teken $\times$ door een andere functie vervangen, al spreekt men dan wel niet meer over vermenigvuldigen. Maar als voorbeeld van de uitbreiding van de matrix en de manier waarop de bewerking gebeurt kan hiermee worden volstaan:

$$\begin{array}{cccc}
 & \leftarrow M \leftarrow 2 & 2p1 & 0 & 0 \\
 1 & 0 & & & \\
 0 & 1 & & & \\
 & \leftarrow N \leftarrow 2 & 3p1 & & \\
 1 & 1 & 1 & & \\
 1 & 1 & 1 & & \\
 & M \circ . = N & & & \\
 1 & 1 & 1 & & \\
 1 & 1 & 1 & & \\
 0 & 0 & 0 & & \\
 0 & 0 & 0 & & \\
 0 & 0 & 0 & & \\
 0 & 0 & 0 & & \\
 1 & 1 & 1 & & \\
 1 & 1 & 1 & &
 \end{array}$$

De vier elementen van de eerste (linkse) matrix zijn vergeleken met de elementen van de tweede (rechtse) matrix. Het resultaat van de vergelijkingen is een logische één als de elementen gelijk zijn, en een nul in het andere geval. Ook bij deze operator moeten de matrices gelijke dimensies hebben.

5.8 Inverteren en delen van matrices

Dit onderwerp is wellicht minder belangrijk voor diegenen die het echte programmeren belangrijk vinden zonder de details van een wiskundige *APL*-functie. De rest van het hoofdstuk kan dan ook zonder verlies van continuïteit worden overgeslagen.

De functie gebruikt voor het delen van matrices wordt ook wel eens de *domino-functie* genoemd om zijn samenstelling van $\square$ en $\div$ tot $\boxdiv$. De functie heeft een monadische en dyadische vorm. In de monadische vorm wordt indien mogelijk de inverse van de matrix berekend. Bij enkelvoudige matrices resulteert dit echter in een domein fout. Twee voorbeelden hieronder tonen aan hoe de domino wordt gebruikt:

```

M←2 2p3 1 2 6
⊞M
0.38 -0.063
-0.13 0.19
A←3 3p19
⊞A
DOMAIN ERROR
⊞A
^

```

In de dyadische vorm wordt de dominofunctie gebruikt bij het oplossen van een stelsel van lineaire vergelijkingen. De gebruikte vorm hiervoor is:

$$B\boxdiv A$$

waarbij V gelijk is aan een vector van één of meerdere dimensies, en M altijd een matrix is. Bij het oplossen van vergelijkingen spreken we normaal over de vorm $aX = b$, met a als de matrix van coëfficiënten van de onbekenden, X de vector onbekenden zelf, en b de vector met de constanten, of waaraan de vergelijking moet voldoen. Door gebruik te maken van de inverse van de matrix a kunnen we de oplossing ook voorstellen als $X = b$ maal inverse a . In *APL* gebeurt dit als:

$$X \leftarrow B\boxdiv A$$

Nemen we als voorbeeld het volgende stelsel van vergelijkingen:

$$\begin{aligned} 2X + Y + Z &= 11 \\ X + 4Y + 2Z &= 24 \\ 3X + Y + 3Z &= 19 \end{aligned}$$

dan gebruiken we de dominofunctie om dit op te lossen als:

$$\begin{aligned} A &\leftarrow 3 \text{ } 3\rho 2 \text{ } 1 \text{ } 1 \text{ } 1 \text{ } 4 \text{ } 2 \text{ } 3 \text{ } 1 \text{ } 3 \\ B &\leftarrow 11 \text{ } 24 \text{ } 19 \\ XYZ &\leftarrow B \oslash A \\ XYZ & \\ 2 \text{ } 4 \text{ } 3 \end{aligned}$$

waar de drie getallen de oplossingen voor X , Y en Z in de vergelijkingen zijn.

5.9 Oefeningen

1. Definiëer een matrix op zo'n manier dat hij er uit ziet zoals de vier voorbeelden hieronder (gebruik een formule of zo min mogelijk elementen als gegevens bij de functie definitie):

10001	$\wedge \wedge \wedge$	$2 \text{ } 4 \text{ } 8$	$\square \begin{smallmatrix} \oplus \\ \oplus \end{smallmatrix} \square$
01010	$\wedge \text{ } \wedge \text{ } \wedge$	$16 \text{ } 32 \text{ } 64$	$\begin{smallmatrix} \oplus \\ \oplus \end{smallmatrix} \square \begin{smallmatrix} \oplus \\ \oplus \end{smallmatrix}$
00100	$\wedge \wedge \text{ } \wedge \wedge$		$\square \begin{smallmatrix} \oplus \\ \oplus \end{smallmatrix} \square$

2. Matrices kunnen worden uitgebreid door er een vector aan toe te voegen zoals bij:

$$\begin{aligned} M &\leftarrow 2 \text{ } 0\rho 0 \\ M &\leftarrow M, [2] \text{ } 1 \text{ } 2 \\ M & \\ 1 & \\ 2 & \\ \rho M & \\ 2 \text{ } 1 & \end{aligned}$$

Gegeven de uiteindelijke vorm en inhoud van de matrix M hierboven, wat is de vorm en inhoud van de matrix na:

$$\begin{aligned} M &\leftarrow M, [2] \text{ } 3 \text{ } 4 \\ M &\leftarrow M, [2] \text{ } 2+, M[; 2] \\ M &\leftarrow M, [1], M[1;] \\ M &\leftarrow M, [1] \text{ } 6 \text{ } 7 \text{ } 8 \end{aligned}$$

3. Gegeven de matrix *PRIVE* uit de tekst. Voer het volgende uit in de context van een budget voor onkosten:
 - a. Bereken de totale onkosten voor VOEDING, KLEDING en ONVOORZIEN.

- b. Bereken het percentage van de onkosten voor TELEFOON in vergelijking met de totale onkosten, en dit voor elke maand afzonderlijk:
 - c. Bepaal de totale onkosten per maand en de procentuele verdeling over de vier maanden.
 - d. Stel een budget samen voor een gemiddelde maand voor elke post, maak een alfanumerieke matrix met de namen van de posten en meldt de naam van die posten die in maand vier meer dan 10 procent van het budget afwijken.
4. Gegeven de matrix met namen van posten uit oefening 3, selecteer de eerste zes karakters, selecteer dan de laatste vier posten en:
- a. Sorteert de matrix in omgekeerde alfabetische volgorde.
 - b. Roteert de matrix rond kolom drie.
 - c. Transponeert de matrix langs de diagonaal.

5. Gegeven de matrix:

$$\begin{pmatrix} 4 & 6 & 3 & 5 \\ 2 & 1 & 9 & 4 \\ 3 & 6 & 2 & 7 \end{pmatrix}$$

Wat is het resultaat na de volgende bewerkingen, en geef aan waar een foutmelding resulteert (verifieer uw antwoord via APL):

$$\begin{array}{l} 2 \ 1 \ \uparrow \ M \\ 2 \ 1 \ \uparrow \ M \\ 4 \ 3 \ \uparrow \ M \\ -2 \ \uparrow \ M \\ -2 \ -2 \ \uparrow \ M \\ 1 \ 0 \ 1 \ / \ M \\ 1 \ 0 \ 1 \ 0 \ / \ M \\ 0 \ 1 \ 0 \ / [1] \ M \end{array}$$

6. Gegeven de matrices A , B , en C :

$$\begin{array}{lll} A: & \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} & B: \begin{pmatrix} 2 & 3 \\ 1 & 2 \\ 4 & 5 \end{pmatrix} & C: \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \end{array}$$

Wat is het resultaat van de volgende bewerkingen en geef aan waar een foutmelding resulteert (verifieer uw antwoord via APL):

$A \times B$
 $A +. \times B$
 $A +. \times B +. \times C$
 $A +. \times C +. \times C$
 $A \wedge. = B$
 $A[;2\ 3] \wedge. = B$
 $A.,[1],C$

7. Schrijf een dyadische functie waarbij de gegevens links en rechts van de functienaam het aantal rijen en kolommen van een matrix zijn. Laat deze functie een patroon tekenen dat er, ongeacht de grootte van de matrix (minimum drie rijen en zes kolommen) als volgt uit ziet:

```

ooo[ ][ ]ooo
ooo[ ][ ]ooo
ooo[ ][ ]ooo
[ ][ ]***[ ][ ]
[ ][ ]***[ ][ ]
[ ][ ]***[ ][ ]

```

8. Schrijf een functie die een stelsel van n vergelijkingen met n onbekenden oplost. Laat alle invoer op een interactieve en gebruiksvriendelijke manier verlopen.
9. Neem uit de tekst de drie-dimensionale matrix A van twee studiegroepjes, elke bestaande uit vier studenten. Sorteert de namen in elke groep en manipuleer de matrix zodanig dat de twee groepen 'naast' elkaar komen te staan, gescheiden door een blanco kolom.
10. Schrijf een functie die een matrix van zeven rijen en vier kolommen transposeert, enkele keren roteert en er een één-dimensionale vector van maakt. De matrix is bepaald in de functie zelf als het alfabet, de punt en de komma. Laat de gebruiker een korte tekst invoeren en gebruik de vector als een codeboek om deze tekst in 'geheime code' om te zetten.

6 Reeksen, logaritmen en goniometrische functies

Zowel in de wiskunde, economie, sociologie als in een aantal andere disciplines is het gebruik van reeksen van getallen met een bepaalde eigenschap nuttig. Welbekend zijn de geometrische reeksen en de goniometrische functies waaronder als bekendste de sinus en de cosinus. Naar gelang de toepassing worden gegevens soms ook op een logaritmische schaal gebracht om een bepaald verloop op een duidelijkere manier weer te geven.

In *APL* kunnen we reeksen van getallen laten genereren door één of andere bewerking met de indexfunctie $\uparrow$. Deze generatie is echter lineair en staat in tegenstelling tot de niet lineaire functies zoals we die in de wiskunde geleerd hebben.

De *APL*-functies in dit hoofdstuk zijn meestal in de dyadische vorm gebruikt om op die manier aan de functie meer gebruiksmogelijkheden te geven. De monadische vorm van deze functies heeft over het algemeen een andere betekenis. De beide vormen worden dan ook bij het aanleren toegelicht.

Om het effect van deze functies te tonen, gebruiken we een gedefinieerde functie die grafische voorstellingen van functies kan geven. We zullen daarom de kennis van de vorige hoofdstukken aanwenden om deze functie te schrijven. Over het algemeen zullen *APL*-gebruikers op een bepaald computersysteem kunnen beschikken over werkgeheugens die uitgebreide grafische faciliteiten bevatten in de vorm van complexe functies. In een goed handboek behorende bij de *APL*-taal van dat computersysteem behoort men dit te kunnen vinden.

Diegenen die bij een eerste lezing van dit boek de te bespreken functies willen overslaan kunnen toch het eerste deel volgen omdat de grafische functie ook in andere hoofdstukken zal worden gebruikt.

6.1 Een gedefinieerde grafische functie

Het ontwerpen van een functie die voor meer dan één vorm van gegevens moet worden gebruikt, vergt over het algemeen meer aandacht dan bijvoorbeeld een functie die alleen een reeks bewerkingen moet uitvoeren voor vaste getallen. Er moet voornamelijk gelet worden op de verschillende mogelijkheden die zich kunnen voordoen. Men kan op verschillende manieren te werk gaan tijdens de voorbereiding op het programmeren. Een eerste manier is het opmaken van een lijst van de bewerkingen, en ook de volgorde van bewerkingen die moeten worden

uitgevoerd. Een blokdiagram vormt een abstracte manier van interpretatie van de lijst en geeft goed aan wat de condities zijn om bepaalde bewerkingen al dan niet te laten uitvoeren. Daarentegen is een structuurdiagram niet meer gebaseerd op verspringen binnen de functie maar berust op de opeenvolging van goed gestructureerde delen van de functie of groepen van berekeningen.

Omdat de grafische functie hier zó gekozen is dat de opeenvolging van bewerkingen op een bijna vanzelfsprekende manier verloopt, zullen we volstaan met een lijst te maken van de te groeperen bewerkingen. De bewerkingen in de functie moeten er voor zorgen dat een reeks waarden kan worden geschetst ten opzichte van een reeks oplopende getallen. Men kan aan de hand van een voorbeeld een functie schrijven die twee vectoren als invoer heeft, een vector voor de x-as en een vector voor de y-as. Als we een matrix definiëren die de rijen en kolommen aangeeft van het te tekenen veld dan rest alleen nog het invullen van deze matrix.

We noemen de gedefinieerde functie *SCHETS* die als doel heeft de waarden van een vector grafisch weer te geven met als referentie de waarden van de x-as. We hebben dus twee invoervectoren nodig, één om de waarden die geschetst moeten worden in op te slaan, en één om als x-as referentie te dienen. Deze vectoren noemen we gemakshalve respectievelijk *Y* en *X*. De definitie van de *APL*-functie is als volgt:

$\nabla X \text{ SCHETS } Y$

Een eerste eis waar de vectoren aan moeten voldoen is, dat zij een gelijke lengte hebben, dus voor elke waarde in *Y* moet een waarde van *X* aangegeven zijn. Wanneer dit niet zo is, moet de functie met een foutmelding komen. We schrijven dus reeds op de eerste regel:

[1] $\rightarrow ((\rho X) \neq \rho Y) / \text{NIETGELIJK}$

om ergens verder in de functie de regels te bepalen die de foutmelding verzorgen:

[...] *NIETGELIJK*: 'LENGTE VAN X EN Y NIET GELIJK' $\rightarrow$ OPNIEUW'
[...] $\rightarrow 0$

Hier merken we op dat bij de verwijzing of het verspringen naar regelnummer 0 betekent dat de functie wordt beëindigd. Men moet in dat geval de lengte van de twee vectoren gelijkstellen en de functie opnieuw gebruiken.

We spreken af dat een grafische voorstelling altijd op een even groot oppervlak plaatsvindt. Arbitrair hebben we hiervoor een maximum hoogte van 20 regels en een maximum breedte van 40 posities gekozen. Aangezien we met vaste afme-

tingen werken, zou dit kunnen betekenen dat we alleen maar waarden mogen gebruiken die voor Y tussen 1 en 20 liggen en voor X tussen 1 en 40. De algemeenschap van de functie moet ons echter toelaten ook andere waarden te gebruiken. Voor Y mogen die zelfs negatief zijn, maar voor X houden we het in deze functie bij positieve waarden.

Om grotere waarden toe te laten zullen we de hele schets op een bepaalde schaal moeten tekenen. Deze schaal kunnen we gemakkelijk berekenen. Aan de hand van drie verschillende reeksen voor Y en een reeks voor X als voorbeeld bespreken we de schaalberekening:

$$\begin{array}{l} Y1 \leftarrow 1 \quad 4 \quad 9 \quad 10 \quad 8 \quad 7 \quad 2 \quad 1 \quad 5 \\ Y2 \leftarrow 1 \quad 2 \quad 4 \quad 8 \quad 16 \quad 32 \quad 64 \quad 128 \quad 64 \quad 16 \\ Y3 \leftarrow 4 \quad 3 \quad 2 \quad 1 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 6 \\ X \leftarrow 10 \end{array}$$

Als we de reeks $Y1$ nemen dan zien we dat de waarden variëren van 1 tot en met 10. Deze waarden kunnen op een hoogte van 1 tot 20 worden geschetst. Hiervoor moet de reeks waarden wat 'uitgerekt' worden, zodat waarde 10 op de hoogste regel wordt afgedrukt, en waarde 1 onderaan. Hiervoor kunnen we de waarden vermenigvuldigen met 2. De schaal is dan 2 op 1. Dit berekenen we door de laagste waarde in de vector te nemen en die af te trekken van de hoogste waarde. Dit resultaat wordt vermeerderd met een tiende deel van de hoogste waarde in Y om latere afrondingsfouten te minimaliseren. In dit voorbeeld krijgen we dan $(10-1)+(10/10)=10$. Als we dit *bereik* uitspreiden over de gestelde limiet van 20 regels dan delen we dit resultaat door 20 (of vermenigvuldigen met .05), en krijgen zodoende 0.5. De waarden in $Y1$ zullen daarna met $1/.5$ of 2 moeten worden vermenigvuldigd, wat inderdaad de schaal aangeeft. In de functie berekenen we dus de schaal van Y als:

$$[2] \quad SCHAALY \leftarrow 1 \div W \leftarrow 0.05 \times ((\uparrow/Y) \div 10) + (\uparrow/Y) - \downarrow/Y$$

We interpretern deze regel van rechts naar links zodat eerst het verschil tussen de maximale en minimale waarde wordt berekend. (Wat gebeurt er als de laagste waarde negatief is?) Dit vermeerderen we met een tiende van de hoogste waarde. Dit nieuw resultaat vermenigvuldigen we met de hoogte factor (.05) en houden dit tussenresultaat vast in de variabele W . De schaal is dan 1 gedeeld door de waarde van W .

Bij de vector $Y2$ volgt een zelfde bewerking. We berekenen eerst het bereik als $(128-1)+12.8=139.8$. De schaal is dus 1 gedeeld door het produkt $139.8 \times .05$ of $1/6.99=.143$. Als we bijvoorbeeld 64 met deze schaalwaarde zouden vermenigvuldigen, krijgen we de waarde 9 afgerond, of een waarde ongeveer te schetsen in het midden van de grafiek.

Voor de vector $Y3$ is deze berekening identiek. Men krijgt dan weliswaar het

bereik van de waarden als een positief getal, en het resultaat van de schaalwaarde is 1.888.

De vector Y passen we nu in de functie aan door een vermenigvuldiging met het resultaat van regel [2]. Omdat er afrondingen kunnen voorkomen tellen we eerst .5 bij de verkregen waarden en ronden dan af naar beneden. Het resultaat van deze bewerking slaan we op in een vector die RIJ heet:

$$[3] \quad RIJ \leftarrow \lfloor .05 + Y \times SCHAALY$$

Voor de vectoren $Y1$, $Y2$ en $Y3$ wordt dit dan respectievelijk:

$$\begin{array}{cccccccc} & Y1 & & & & & & \\ 2 & 8 & 18 & 29 & 16 & 14 & 6 & 4 & 2 & 10 \\ & Y2 & & & & & & & & \\ 0 & 0 & 1 & 1 & 2 & 5 & 9 & 18 & 9 & 2 \\ & Y3 & & & & & & & & \\ -8 & -6 & -4 & -2 & 0 & 2 & 4 & 6 & 8 & 11 \end{array}$$

Bij de eerste vector is er geen enkel probleem en de te schetsen waarden vallen netjes tussen de waarden 1 en 20. Bij de tweede vector zijn er echter een paar waarden gelijk aan nul. Als we de rij-index zullen moeten gebruiken voor de plaatsbepaling in een matrix zal dit een indexfout geven. We zullen deze nullen omvormen tot één nadat we de schets aangepast hebben voor de negatieve waarden. Het is duidelijk dat ook negatieve indicering tot fouten zal leiden. Daarom moeten we de waarden in de variabele RIJ vergroten met het grootste negatieve getal +1. Omdat de te schrijven APL -instructie ook voor die gevallen moet werken waar geen negatieve waarden voorkomen, voeren we eerst een test in om te zien of er negatieve getallen voorkomen. Is dit niet zo, dan wordt er nul bij de variabele opgeteld. Indien dit wel zo is wordt er de kleinste waarde van de rij-index bijgeteld. Dit kunnen we doen met de volgende instructie:

$$[4] \quad RIJ \leftarrow RIJ + (1 + \lfloor \lfloor RIJ \rfloor \times 0 \geq \lfloor RIJ$$

Voor de vectoren $Y1$ en $Y2$ blijft het resultaat ongewijzigd. Voor vector $Y3$ wordt dit:

$$\begin{array}{cccccccc} & RIJ & & & & & & \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 20 \end{array}$$

Als er na deze bewerking nog een nul zou voorkomen dan kunnen we ter voorkoming van een indexfout, de waarde één toewijzen. Indien door afronding de waarde 21 zou voorkomen, dan moet deze tot 20 worden verlaagd. Dat doen we als volgt:

$$[5] \quad RIJ \leftarrow RIJ + 0 = RIJ \leftarrow RIJ + (20 < RIJ) \times^{-1} 1$$

De logische vector met een waarde één wordt vermenigvuldigd met -1 op die plaatsen waar 21 voorkomt. Deze vector wordt van de rij-index afgetrokken zodat er op de juiste plaatsen 20 komt te staan. Daarna wordt de logische vector berekend met een één waar nul stond. Deze vector wordt dan per corresponderend element bij de rij-index geteld. Dit alles heeft tot gevolg dat voor de vector $Y2$ de waarden zijn geworden:

RIJ
1 1 2 2 3 6 10 19 10 3

De berekeningen hierboven zijn ook van toepassing op de vector X waar het resultaat van de schaal aanpassing in de variabele $KOLOM$ opgeslagen wordt. We veronderstellen hier dat X een vector is met toenemende waarden, meestal oplopend van 1 tot en met ρX . De schaal aanpassing verloopt eenvoudiger en de afrondingsfouten zijn dan niet zo groot. De gedefinieerde functie ziet er nu voorlopig als volgt uit:

$\forall X \text{ SCHETS } Y$

```
[1]  →((ρX)≠ρY)/NIETΔGELIJK
[2]  SCHAALY+1÷W+0.05×((Γ/Y)÷10)+(Γ/Y)-L/Y
[3]  RIJ+L 0.05+Y×SCHAALY
[4]  RIJ←RIJ+(1+|L/RIJ)×0≥L/RIJ
[5]  RIJ←RIJ+0=RIJ←RIJ+(20<RIJ)×-11
[6]  SCHAALX+1÷0.025×1+(Γ/X)-L/X
[7]  KOLOM+L 0.5+X×SCHAALX
[8]  KOLOM←KOLOM+0=KOLOM
```

Bij het einde van de berekening van regel [8] zijn bij het gebruik van de vector $Y1$ als Y de waarden van de rij- en de kolom-indicering:

RIJ
2 8 18 20 16 14 6 4 2 10
 $KOLOM$
2 4 6 8 10 12 14 16 18 20

Deze resultaten interpreteren we als volgt. De eerste waarde in de rij-index en de eerste waarde in de kolom-index geven de positie weer in een 20×40 matrix, waar een waarde moet worden getekend. In dit geval is het dus in rij twee en kolom twee. Aangezien we echter de rijen in een matrix van boven naar beneden nummeren, is rij twee bovenaan, terwijl we in een grafische voorstelling de tweede rij onderaan bedoelen. Bij de opvulling van de matrix zullen we daarom de rij-index van het getal 21 aftrekken. Om de matrix te definiëren en de te tekenen waarden aan te geven, maken we gebruik van een ingebouwde herhaling van instructies.

Dit wordt in het Engels *loop* genoemd. We voeren hiermee voor elk van de kolommen de opvulling van de matrix als:

```
[9]      M← 20 40 p' '
[10]     I←0
[11]     TERUG:→((pX)<I+I+1)/TEKEN
[12]     M[21-RIJ[I];KOLOM[I]]←'o'
[13]     →TERUG
```

Op regel [11] testen we of alle waarden reeds getekend zijn. Dit doen we door te testen of de indexvariabele I niet groter wordt dan de lengte van de vector X . We hadden ook op de lengte van vector Y kunnen testen, maar X geeft een probleem indien er meer dan 40 waarden worden opgegeven. Het kan namelijk zijn dat voor een aantal kolommen er twee rij-indexen zijn, omdat de x-as 'samengeduwd' is tot 40 posities. Omdat deze functie slechts een voorbeeld is en gebruikt wordt om de verdere primitieve APL-functies te schetsen, zullen we de lengte van de vectoren kleiner of gelijk aan 40 houden.

Er resteert nu nog het afdrukken van de matrix. Dit kunnen we doen door eenvoudig op de volgende regel M te schrijven, maar om de gebruiker van de gedefiniëerde functie een indruk te geven van wat de geschetste waarden zijn, drukken we de matrix rij per rij af en geven de waarde mee op het einde van elke rij. Daartoe gebruiken we de in regel [2] berekende waarde van W (de grootte van een twinstigste deel!) en trekken die per rij een aantal keren af van de maximale waarde in de vector Y . De rij zelf laten we voorafgaan door een rechte lijn om het geheel wat meer vorm te geven. Het afdrukken kunnen we als volgt programmeren:

```
[14]     TEKEN:I←0
[15]     PERARIJ:→(20<I+I+1)/EINDE
[16]     ' | ';M[I;];(⌈/Y)-W×I-1
[17]     →PERARIJ
[18]     NIETΔGELIJK:'LENGTE VAN X EN Y NIET GELIJK → OPNIEUW'
[19]     →0
[20]     EINDE:' ';40p'----| '
[21]     (⌊/X);(38p' ');⌈/X
```

Na van het schetsen van de waarden, kunnen we de x-as ook nog in beeld brengen. We trekken eerst een lijn onder de grafiek en drukken dan de laagste (eerste) waarde en de hoogste (laatste) waarde van X af. De functie is nu geschreven en ziet er in zijn totaliteit uit als:

```
▽SCHETS[□]▽
▽ X SCHETS Y;SCHAALX;SCHAALY;RIJ;KOLOM;I
[1] →((pX)≠pY)/NIETΔGELIJK
[2] SCHAALY←1÷W←0.05×((⌈/Y)÷10)+(⌈/Y)-⌊/Y
[3] RIJ←⌊0.05+Y×SCHAALY
[4] RIJ←RIJ+(1+⌊⌊/RIJ)×0≥⌊/RIJ
```



```

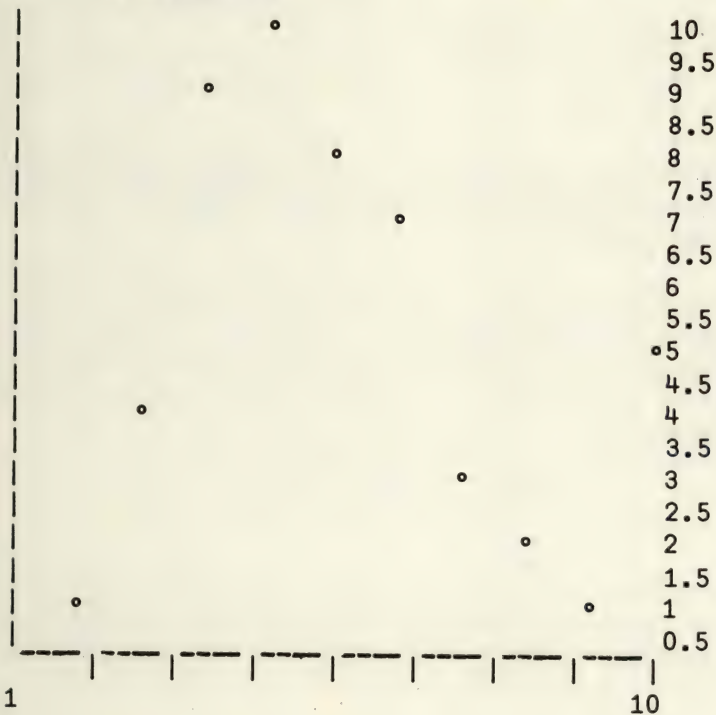
[5]    RIJ←RIJ+0=RIJ+RIJ+(20<RIJ)×-1
[6]    SCHAAALX←1÷0.025×1+(⌈/X)-⌊/X
[7]    KOLOM←⌊0.5+X×SCHAAALX
[8]    KOLOM←KOLOM+0=KOLOM
[9]    M←20 40 p' '
[10]   I←0
[11]   TERUG:→((pX)<I+I+1)/TEKEN
[12]   M[21-RIJ[I];KOLOM[I]]←'o'
[13]   →TERUG
[14]   TEKEN:I←0
[15]   PERARIJ:→(20<I+I+1)/EINDE
[16]   'I';M[I;];(⌈/Y)-W×I-1
[17]   →PERARIJ
[18]   NIETΔGELIJK:'LENGTE VAN X EN Y NIET GELIJK → OPNIEUW'
[19]   →0
[20]   EINDE:' ' ;40p'-----|'
[21]   (⌊/X);(38p' ' );⌈/X

```

▽

Als toepassing nemen we de vector *Y1* als invoer van deze functie en de vector *X*:

X SCHETS *Y1*



Met behulp van deze functie zullen we de reeksen en de continue functies beter kunnen voorstellen. Hierbij zullen de berekeningen zoals de rij-index, schaalwaarden, enz. niet worden afgedrukt. De lezer kan deze functie overnemen en eventueel de nodige veranderingen inbrengen om bepaalde gegevens alsnog bij de uitvoer te stoppen. Het bepalen van globale en lokale variabelen kan dan naar willekeur geschieden.

6.2 Combinatie- en faculteitsfuncties

Voor de faculteit gebruikt men in de wiskunde het uitroepteken á na een variabele of getal (bijvoorbeeld $N! = 1 \times 2 \times 3 \times \dots \times N$). In *APL* staat dit teken vóór een variabele of getal. Het teken zelf wordt gevormd door de punt (.) en het aanhalings-teken ('). Het resultaat geeft het produkt in de monadische vorm weer van $1 \times 2 \times 3 \times \dots \times N$, waar N het rechtse gegeven van de functie vormt:

$$\begin{array}{c} !4 \\ 24 \end{array}$$

Het rechtse gegeven mag echter ook een vector of een decimaal getal zijn zoals te zien is in de volgende voorbeelden:

$$\begin{array}{c} !4 \ 6 \ 9 \\ 24 \ 720 \ 362880 \\ ! \ .24 \\ 0.90852 \end{array}$$

In het laatste geval is het resultaat niet de uitkomst van een vermenigvuldiging $.01 \times 2 \times \dots \times 0.25$, maar de waarde van de gamma-functie zoals die in de wiskunde in de integrale vorm bekend is.

Meer van toepassing lijkt het gebruik van de functie ! in de dyadische vorm. Veronderstel dat bij een schriftelijk tentamen elke student tien vragen krijgt. Daarvan mag de student er vijf naar eigen keuze oplossen. De docent krijgt dan heel wat combinaties van tentamens. Het maximum aantal dat de docent kan krijgen is bepaald door:

$$\begin{array}{c} 5!10 \\ 252 \end{array}$$

of het resultaat van het aantal combinaties van vijf uit tien. Ook in de dyadische vorm mag een vector gebruikt worden, zowel rechts, links of aan beide kanten van de functie. Dit geeft dan volgens de lengte van de vector de volgende resultaten:


```

          3! 3 5 6
1 10 20
          2 3 ! 5
10 10
          3 4 ! 5 7
10 35

```

Wanneer twee vectoren gebruikt worden moeten beide dezelfde lengte hebben wil men een domein-fout vermijden. De berekening vindt dan wel plaats met de corresponderende elementen in de vectoren.

6.3 De circelfunctie en logaritmen

De circelfunctie wordt gekenmerkt door het gebruik van het kleine circeltje dat boven de letter *O* op het toetsenbord te vinden is. Deze *APL*-functie $\circ$ is zeer belangrijk bij vele wiskundige bewerkingen, vooral die welke te maken hebben met de meetkunde. In de monadische vorm is de circelfunctie te gebruiken als de constante π in de eenvoudigste vorm, namelijk als $\circ 1$. Het getal rechts van de functie geeft aan hoeveel maal men het getal π wil hebben, zoals in:

```

          ○1
3.1416
          ○2
6.2832
          ○5
15.708

```

Om bijvoorbeeld de omtrek van een cirkel te berekenen als $2 \times r \times \pi$ waar r de straal is, schrijven we in *APL*:

```

          R←3
          R×○2
18.85

```

Een andere toepassing van de circelfunctie in de monadische vorm is het berekenen van graden of radialen. Als een hoek gegeven is in graden en men radialen moet gebruiken, dan is de conversie uit te voeren als:

```

          GRADEN←30 45 60
          GRADEN×○÷180
0.5236 0.7854 1.0472

```

waar ook bij de omzetting een vector mag worden gebruikt. De omgekeerde conversie verloopt dan als:

$$\begin{aligned} \text{RADIALEN} &\leftarrow .34 .65 \\ \text{RADIALEN} &\times 180 \div 01 \\ 19.481 \quad 37.242 \end{aligned}$$

Bij de circelfuncties sinus, cosinus, enz. moeten de gegevens in radialen zijn. De conversie zal dus van pas komen bij de bespreking van deze goniometrische functies.

Ook het gebruik van logaritmen komt heel dikwijls in allerlei toepassingen voor. *APL* heeft een primitieve functie of functie waarmee de berekening van logaritmen op een eenvoudige manier kan. Omdat het gebruik van de circelfunctie samen gebruikt wordt met de exponentfunctie (dus ook niet-lineair), bespreken we hier de logaritme-functie. De functie wordt gevormd door de samenstelling van $\circ$ en $*$ die naast elkaar op het toetsenbord te vinden zijn. De functie wordt in de monadische en dyadische vorm gebruikt. In het eerste geval om de natuurlijke logaritme van een getal te berekenen zoals in:

$$\begin{aligned} & \circ 2 \\ 0.69315 \\ & \circ 4 \quad 5 \quad 6 \\ 1.3863 \quad 1.6094 \quad 1.7918 \end{aligned}$$

In de dyadische vorm gebruiken we de logaritme-functie om de logaritme volgens een bepaalde basis te berekenen. Hierbij wordt links van de functie de basis gegeven, en rechts ervan het getal in kwestie. De voorbeelden spreken voor zichzelf, alsook de interpretatie in monadische vorm:

$$\begin{aligned} & 5 \circ 2 \\ 0.43068 \\ & (\circ 2) \div \circ 5 \\ 0.43068 \end{aligned}$$

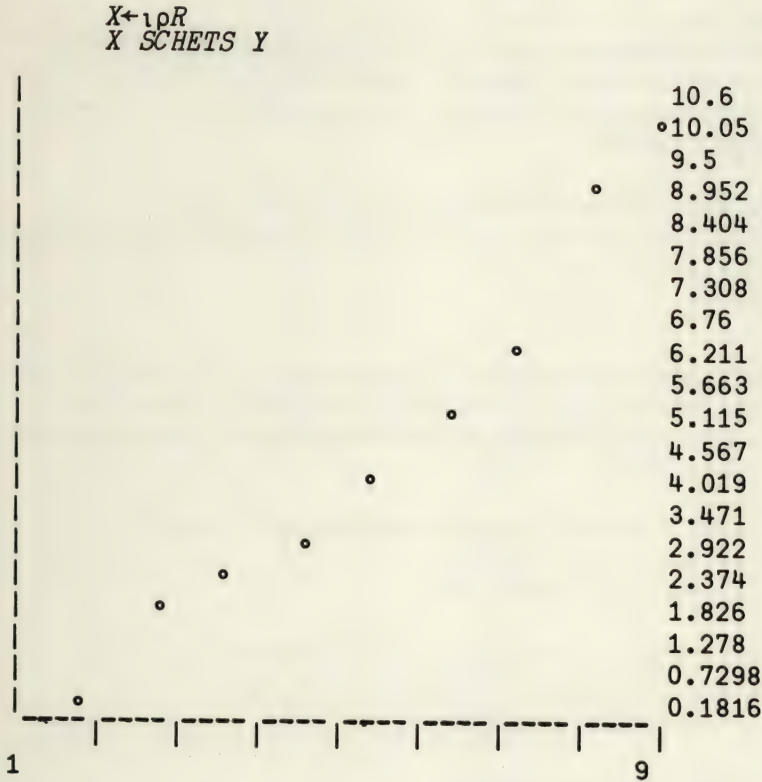
Hierbij moet de opmerking gemaakt worden dat zowel het linkse en/of rechtse gegeven van de functie groter dan nul moeten zijn en dat de basis, dus het linkse getal, niet gelijk mag zijn aan één, tenzij het rechtse getal ook één is.

Als toepassing bespreken we een reeks getallen die een dynamische reeks gegevens voorstellen. Deze reeks ziet er als volgt uit:

$$R \leftarrow 2 \quad 9 \quad 25 \quad 36 \quad 100 \quad 400 \quad 1020 \quad 11000 \quad 40000$$

Als we deze reeks als een functie willen schetsen dan lijkt dit een enorm werk zelfs op een zeer groot stuk papier. De logaritmeberekening kan ons hierbij helpen omdat we de reeks dan op een logaritmeschaal kunnen weergeven. We tonen dit door eerst de natuurlijke logaritme van de reeks te gebruiken en dan de nieuwe vector met behulp van de functie *SCHETS* grafisch voor te stellen:

$\square \leftarrow Y \leftarrow \odot R$
0.693147 2.19722 3.21888 3.58352 4.60517 5.99146 6.92756
9.30565 10.5966



6.4 Goniometrische functies

We hebben reeds gezien dat de circelfunctie $\odot$ gebruikt wordt voor continue functies. In de dyadische vorm vormt de functie afhankelijk van het linkse getal een bepaalde functie. Deze zijn hierna samengevat:

SINUSOIDALE FUNCTIE

1	○	SINUS
2	○	COSINUS
3	○	TANGENT
5	○	HYPERBOLISCHE SINUS
6	○	HYPERBOLISCHE COSINUS
7	○	HYPERBOLISCHE TANGENT

Deze lijst kan worden aangevuld met de negatieve waarden van de linkse gegevens. Zodoende krijgt men de berekening van de boogsinus, boogcosinus, enz.. Uit deze lijst zijn de linkse gegevens 1, 2 en 3 (alsook -1, -2 en -3) de meest gebruikte. Het volgende voorbeeld geeft het gebruik weer.

Veronderstel dat een ladder met een lengte van vijf meter zodanig tegen een muur staat, dat de ladder er onderaan anderhalve meter vanaf staat. Op een tentamen wordt dan meestal gevraagd wat de hoek is waar de ladder en de muur elkaar raken. Dit lossen we op als:

A HOEK IS BOOG SINUS VAN DE AANLIGGENDE
 A ZIJDE GEDEELD DOOT DE TEGENOVERGESTELDE ZIJDE
 $\square \leftarrow \text{HOEK} \leftarrow 1.05 \div 1.5$

0.30469

Het resultaat is nu in radialen uitgedrukt. Om het probleem nog wat uit te breiden zouden we nu kunnen vragen hoe ver de ladder van de muur komt te staan als we de gevonden hoek met ongeveer drie graden zouden vergroten. Deze berekening laten we als volgt verlopen:

A DE 3 GRADEN WORDEN OMGEZET NAAR RADIALEN EN
 A BIJ DE HOEK GETELD.
 $N\Delta \text{HOEK} \leftarrow \text{HOEK} + (0.3) \div 180$

A AFSTAND IS DE TEGENOVERGESTELDE ZIJDE OF
 A DE SINUS VAN DE NIEUWE HOEK MAAL D
 A AANLIGGENDE ZIJDE

$\square \leftarrow \text{AFSTAND} \leftarrow 5 \times 1.0 \times N\Delta \text{HOEK}$

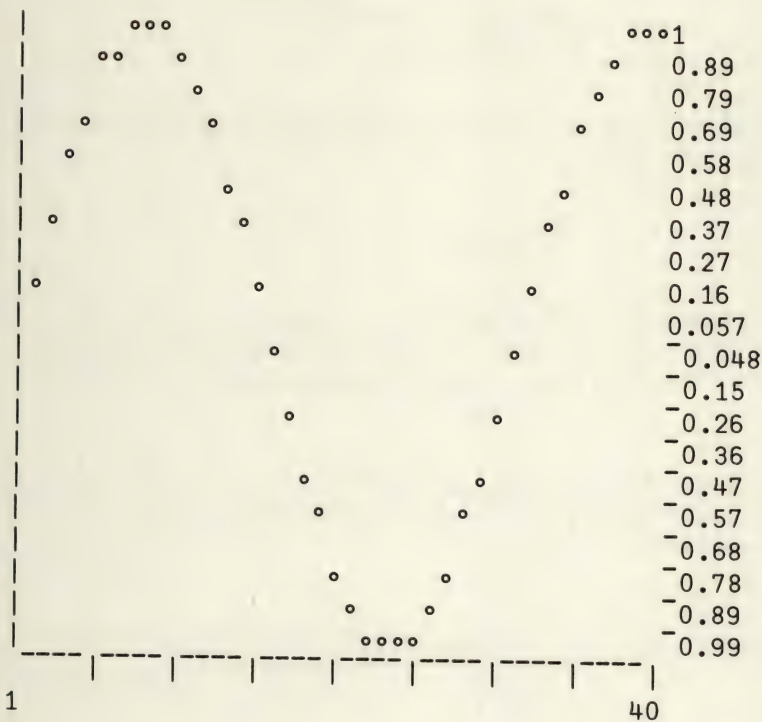
1.7476

Hierbij hebben we dus eerst het aantal graden dat de hoek groter wordt, omgezet in radialen. Dit opgeteld bij de beginwaarde van de hoek dient nu als rechtse gegeven voor de sinusfunctie. De lengte van de tegenovergestelde zijde, of de afstand van de muur tot de ladder, wordt dan berekend als de uitkomst van de sinuswaarde maal de lengte van de ladder (de aanliggende zijde.)

Een ander voorbeeld betreft het genereren van een sinusoïde. We nemen daartoe een vector van verschillende waarden rechts van de functie zodat voor elke waarde de sinus berekend kan worden. De berekende waarden schetsen we met behulp van de functie uit dit hoofdstuk. De x-as en de y-waarden worden als volgt bepaald:

$$\begin{aligned} X &\leftarrow 140 \\ Y &\leftarrow 1 \circ X \div 5 \\ X \text{ SCHETS } Y \end{aligned}$$

De y-waarden in de vector S zijn tot 8 significante cijfers berekend. We hebben de getallen afgerond zodat de schets er als volgt uitziet:



De afrondingen in de functie hebben ertoe geleid dat boven en onder in de schets de waarden wat ineengedrukt zijn. De afvlakking van de sinuslijn bij de waarden 1 en -1 is door het schetsen volgens een benadering van een continue functie wat sterker overgekomen. Het voornaamste is echter dat een indruk kan worden gegeven van het verloop van een functie. In de oefeningen kunnen eventueel nog andere functies geschetst worden.

6.5 Oefeningen

1. Bereken met de hand het resultaat voor de volgende bewerkingen:

$$\begin{array}{r} !1 \\ !5 \\ !1 \ 2 \ 3 \\ 2 \ ! \ 5 \\ \circ 1 \end{array}$$

2. Wat is het resultaat van de volgende bewerking? Indien een fout resulteert, verklaar deze dan.

$$\begin{array}{l} X \leftarrow 15 \\ Y \leftarrow 1 \circ X \div 1 \ 2 \ 3 \ 4 \ 5 \\ X \text{ SCHETS } Y \end{array}$$

3. Wat is het resultaat van de volgende bewerking? Indien een fout resulteert, verklaar deze dan.

$$\begin{array}{r} 70 \\ -8 \circ 8 \end{array}$$

4. Schrijf een functie die op interactieve basis de gegevens opvraagt voor de grafische functie en die dan, na het controleren van de gegevens, de gewenste schets maakt. Eventueel kan de gebruiker gemeld worden wat de X en Y schalen zijn.
5. Bereken 40 waarden van de cosinus-functie tussen -1 en 1 , en 40 waarden van de tangens tussen -0.5 en $.05$. Schets met behulp van de grafische functie de bereikte resultaten.

7 Invoer- en uitvoerproblematiek

Zoals we *APL* tot nu toe als calculator of als computer via de gedefinieerde functies hebben gebruikt, kregen we invoer en uitvoer van gegevens en resultaten zonder enige vorm van 'formaat'. We bedoelen hiermee dat de gegevens altijd in een vrije vorm gebruikt werden zodat getallen die tot tien significante cijfers waren berekend, ook zo werden afgedrukt. Bij alfanumerieke tekst was de uitvoer wel enigszins geformatteerd omdat alleen de vector zelf werd afgedrukt.

Bij de invoer van gegevens praat men bij *APL* alleen maar over de logische volgorde van het invoeren van gegevens. Bij de uitvoer daarentegen zal men tevoren willen bepalen waar bijvoorbeeld getallen moeten staan, zoals in een rapport, en hoeveel cijfers er voor en na de decimale punt moeten staan. Er zijn in *APL* verschillende manieren om te formatteren, ieder met zijn voor en nadelen.

Een ander facet van het afdrukken van gegevens is het combineren van oewel numerieke als alfanumerieke resultaten. In de gedefinieerde functies die we tot nu toe gezien hebben zou zo'n combinatie reeds wenselijk zijn geweest. Vooral bij de functie in hoofdstuk 6 waarmee we reeksen grafisch kunnen weergeven, hebben we gemerkt dat de uitvoer er nogal ongeformatteerd uitziet. Zonder veel in deze functie te wijzigen zullen we er een kleine verandering in aanbrengen om tot een beter resultaat te komen.

7.1 Invoer buiten een gedefinieerde functie

Aan de hand van een reeks voorbeelden en verschillende oefeningen hebben we opgemerkt dat het invoeren van gegevens ook buiten de gedefinieerde functies kan gebeuren. In de meeste gevallen werd gebruik gemaakt van de eenvoudige vormen:

```
V← 1 4 6 8 , 15
A← 'DIT IS EEN TEKST'
```

voor numerieke en alfanumerieke gegevens. Men kan echter ook het *quad* teken □ gebruiken om dan vervolgens de gegevens in te typen. Hierbij mag men bij het gebruiken van □ ook alfanumerieke gegevens invullen, mits ze tussen aanhalingstekens (') geplaatst zijn. Dit laatste kan men ook zonder aanhalingstekens doen

met het invoerteken $\square$. Beide tekens mogen ook gebruikt worden voor gegevens op een volgende regel, zoals in:

```

      V← $\square$ 
 $\square$ :
      1 2 3 4 5, $\square$ 
 $\square$ :
      6 7 8 9 10
      V
1 2 3 4 5 6 7 8 9 10
      A←'DE VRAAG IS ', $\square$ 
      REEDS GESTELD
      A
      DE VRAAG IS REEDS GESTELD

```

Het verschil tussen beide tekens is dat bij invoer van numerieke gegevens de *quad* wordt afgedrukt gevolgd door een dubbele punt. Bij de *quad quote* kunnen reeds op de direct volgende regel gegevens worden ingevoerd. In dit laatste geval zijn geen aanhalingstekens nodig.

7.2 Invoer binnen een gedefinieerde functie

Net als berekeningen geldt dat invoer binnen een functie hetzelfde is als invoer buiten een functie. Alleen worden hier nog een paar varianten gebruikt zoals:

```

      KEUZE← $\square$ 

      →(10< $\square$ )/TIEN

      TOTAAL←BEDRAG×(100+ $\square$ )÷100

```

In het eerste voorbeeld wordt een scalaire of een vector als antwoord verwacht. In het algemeen mag dit ook een globale variabele zijn die als scalaire of vector gedefinieerd is. Dit geldt trouwens ook voor invoer buiten de functie om. Eventuele bewerkingen zoals 3×7 kunnen ook als invoer van het getal 21 dienen. In het tweede voorbeeld wordt een beslissing in de functie genomen in de vorm van een test op het ingevoerde getal. Als tijdens de uitvoering van de functie om een getal gevraagd wordt, en dit getal is groter dan tien, dan gaat de uitvoering verder met de regel die de naam *TIEN* heeft meegekregen. Is het getal kleiner of gelijk aan tien, dan wordt de volgende regel in de functie uitgevoerd. Het derde voorbeeld geeft aan dat er invoer mogelijk is tijdens de berekeningen zelf. In dit geval kan er een BTW-percentages worden ingevoerd. Indien daarvoor het getal 18 wordt gebruikt, dan wordt het bedrag vermenigvuldigd met $118/100$ of 1.18.

Tijdens het uitvoeren van een functie komt het vaak voor dat de functie moet worden gestopt. Men kan dit over het algemeen wel doen met een indrukken van de ATTENTION-toets, maar dit kan soms tot gevolg hebben dat er geen controle is waar nu precies de functie stopt. Vervelender is het feit dat als er invoer van de gebruiker van de *APL*-functie verwacht wordt, de functie dikwijls nog niet op dat punt is gestopt omdat het na het indrukken van de ATTENTION toets nog eventjes duurt. Als de plaats van invoer bereikt wordt vóórdat er gestopt is, dan heeft de gevraagde onderbreking geen enkel effect. De functie wacht dus op verdere invoer. Toch kan men de functie stoppen door op het moment van invoer het woord *OUT* in te typen, zodanig dat de drie letters op elkaar komen te staan. Dit kan door de BACKSPACE-toets te gebruiken. De functie stopt dan met het bericht dat op die lijn de functie onderbroken is. Dit gebeurt als volgt:

```

      ∇TEST
[1]  K←10
[2]  K←K,∇
[3]  →2×25>pK ∇

```

```

      TEST
WE TONEN HIER
HET GEBRUIK VAN
∇
INTERRUPT
TEST[2]  K←K,∇
      ^

```

De functie stopte dus op regel [2] en bij de meeste *APL*-systemen wordt de exacte plaats aangeduid met het teken $\wedge$. Dezelfde melding komt voor bij een functie zoals *SCHETS* waar geen invoer wordt verwacht en waar dus wel de ATTENTION-toets (bij enkele terminals is dat de BREAK-toets) wordt gebruikt:

```

      X SCHETS Y
|      °°°
|      °° °
|      °
|
|      °°°1
|      ° 0.89
|      ° 0.79
INTERRUPT
SCHETS[16] ' '|;M[I;];(I/Y)-W×I-1
      ^

```

In beide laatste voorbeelden kan men alsnog de functie vervolgen door zelf een regelverwijzing te gebruiken zoals binnen de functie, namelijk $\rightarrow 2$, zodat voor de functie *TEST* de executie vervolgd wordt tot er meer dan 25 karakters ingevoerd zijn:

```

      →2
DE FUNCTIE KAN NU EINDIGEN

```

Hier is een belangrijke opmerking nodig. Bij het vervolgen van een onderbroken functie door het gebruik maken van de verwijzing naar de regel, is het noodzakelijk te weten tijdens welke functie het laatst werd gestopt. De verwijzing geldt immers alleen voor deze functie. Dus als we ook nog de grafische functie om te schetsen hadden gestopt, dan was het resultaat van de verwijzing een voortzetting geweest vanaf regel [2] in de functie *SCHETS* !

Bij invoer van numerieke gegevens kan men de functie laten stoppen en meteen ook beëindigen door in plaats van getallen in te voeren, het verwijzingsteken $\rightarrow$ in zijn niladische vorm te gebruiken. Op dat moment kan geen voortzetting van de onderbroken functie meer geschieden. Deze techniek werkt niet voor die gevallen waar het teken $\square$ gebruikt werd voor invoer.

Een laatste opmerking betreffende de invoer van gegevens tijdens de uitvoering van een functie houdt verband met de controle op de hoeveelheid van gegevens. In de veronderstelling dat de door de programmeur geschreven functie ook nog wel eens door anderen zal worden gebruikt, zal het verschijnen van de tekens $\square$: tijdens de executie wellicht vragen oproepen in de geest van, wat moet hier worden ingevoerd, of, hoeveel getallen zijn hier nodig. Men doet er goed aan bij het schrijven van een functie vóór elke invoer een tekst te laten afdrukken of op het scherm te laten verschijnen. Deze tekst kan eventueel uitleg geven over het type van gegevens dat moet worden ingetypt, en/of het aantal gegevens. Na de invoer zelf is het altijd raadzaam na te gaan of aan de instructies voldaan is. Het invoeren van bijvoorbeeld een vector in plaats van een scalaire, kan ongewenste gevolgen hebben in het verdere verloop van de functie.

7.3 Uitvoer zonder formaat

Voor de uitvoerproblematiek maken we geen onderscheid tussen het weergeven van resultaten binnen of buiten de werking van een functie. De eenvoudigste vorm van uitvoer hebben we reeds in de vorige hoofdstukken toegepast, zoals door het typen van de variabelenaam of een bewerking buiten de functie, of op een regel binnen de functie. Nemen we bijvoorbeeld de vierkantswortel uit elf dan krijgen we het berekende resultaat:

$$11 \star .5$$

$$3.31662479$$

Het totaal aantal significante cijfers wordt in het werkgeheugen bepaald door een systeemvariabele. Dit soort variabelen kunnen desgewenst naar wens worden veranderd, maar hebben bij het eerste gebruik van een werkgeheugen een bepaalde waarde meegekregen. De interactie van de gebruiker met deze systeemvariabelen wordt uitvoerig in hoofdstuk 9 besproken. We behandelen hier de systeemvariabele die het aantal cijfers bepaalt. Het aantal is in principe tien. Een ver-

andering met het commando `)DIGITS 2` en het resultaat hiervan ziet er als volgt uit:

```

)DIGITS 2
WAS 10
      11*.5
      3.3

```

Het resultaat is dus een verandering van het aantal cijfers, met het gevolg dat er een 'afronding' heeft plaatsgevonden bij het afgedrukte resultaat. Als we bijvoorbeeld de sinuswaarden in hoofdstuk 6 wat korter zouden willen hebben, dan kunnen we van deze verandering van de systeemvariabele gebruik maken. Dit is echter niet aan te raden omdat er voor verschillende soorten resultaten verschillende aantallen significante cijfers kunnen zijn. In *APL* zullen we daarom formaten gebruiken die per omstandigheid de gewenste afdrucken geven.

De uitvoer van een karaktervector hoeft niet door middel van een formaat plaats te vinden. Er kan volstaan worden de vector als instructie op een regel te plaatsen, of de naam van de vector op te geven. Eventueel kan een vector nog met één of meerdere karakter of alfanumerieke vectoren worden samengevoegd. Bijvoorbeeld:

```

A←'DRUK DEZE ZIN AF'
A
DRUK DEZE ZIN AF
[←A
DRUK DEZE ZIN AF
'LET OP: ',A
LET OP: DRUK DEZE ZIN AF

```

In het laatste voorbeeld vond de samenvoeging plaats met een komma als functie. Dit mogen we alleen doen als het type vector en de rangorde van de vectoren gelijk is.

Eerder hebben we reeds opgemerkt dat het gebruik van numerieke en alfanumerieke gegevens voor uitvoer met enige zorg moet worden gedaan. In enkele gedefiniëerde functies hebben we bijvoorbeeld gebruik gemaakt van de punt-komma. Maar de afgedrukte regel is nog geen alfanumerieke of numerieke vector en kan later ook niet voor verwerking worden gebruikt. Er is echter in *APL* een manier om van het numerieke gedeelte van uitvoer ook een alfanumeriek equivalent te maken. Dit lichten we even toe met een voorbeeld.

Om een getal om te zetten naar een karakterequivalent, kunnen we op een ongeformatteerde wijze gebruik maken van het toewijzingsteken `←`. Het resultaat van de bewerking `⌈←?9` is een karaktervector met lengte één. Deze vector die

volgens de uitvoer slechts een numeriek gegeven bevat kan evenwel niet meer worden gebruikt om er een numerieke bewerking mee uit te voeren. Dit blijkt uit het volgende:

```

      A←⍒←'4'
      ρA
1      A
4      A×3
      DOMAIN ERROR
      A×3
      ^

```

In een functie kunnen we hier op een handige manier gebruik van maken om numerieke gegevens samen met alfanumerieke gegevens af te drukken als één enkele karaktervector. Het koppelen van deze beide types van gegevens wordt verder in dit hoofdstuk besproken.

7.4 Uitvoer met formaat

Het gebruik van de systeemvariabele voor het naar wens bepalen van het aantal significante cijfers is in de meeste gevallen erg onhandig. Men heeft *APL* dan ook in de loop der jaren verder ontwikkeld, wat ook voor de uitvoer enkele gevolgen had. Doordat de *APL*-taal meer en meer toepasbaar werd geacht buiten de universitaire en wiskundige sfeer, werd aandacht besteed aan de vorm van uitvoer zoals bij het aanmaken van rapporten waar alles netjes op een rij of kolom moet staan. Hierbij werd de mogelijkheid geboden het aantal significante cijfers voor en na het decimaal teken te specificeren.

Het hiervoor gekozen samengestelde teken wordt gevormd door de tekens $\top$ en $^{\circ}$, boven op elkaar. De volgorde van schrijven is van geen belang zolang de BACKSPACE-toets gebruikt wordt na het eerste karakter. De meest voorkomende vorm van deze functie is de dyadische. Daarbij is het rechtse gegeven een scalaire of een vector dat in het formaat moet worden gezet, bepaald door de scalaire of vector links van de functie.

We nemen als voorbeeld een matrix M die er als volgt uitziet:

```

      M
      1.2E0    3.5E0
      -1.4E1    2.0E2

```

Als we het formaat voor de uitvoer met een vector van twee getallen bepalen, dan volgen we twee regels. De eerste regel betreft het eerste getal in de vector. Dit getal

moet aangeven over hoeveel kolommen de getallen dienen te worden afgedrukt, en wel zo dat er rekening wordt gehouden met het aantal cijfers na de decimale punt, en de punt zelf. Met het aantal kolommen wordt hier het aantal afdruposities bedoeld dat voor de getallen is gereserveerd. Als we bijvoorbeeld naar $M[2;2]$ kijken dan hebben we in totaal vijf posities nodig. Voor het getal 3.4567 zijn er echter zes kolommen nodig wil men alle significante cijfers zien. Men moet echter niet alleen rekening houden met de grootte van de te formatteren getallen, maar ook met het feit of ze negatief zijn. Het negatief teken - moet dan immers ook worden afgedrukt en neemt dus ook een positie in.

De tweede regel betreft het tweede getal in de vector. Dit getal moet aangeven hoeveel significante cijfers na de decimale punt moeten worden afgedrukt, en wel zo dat er rekening mee moet worden gehouden dat getallen op die manier afgerond kunnen worden. Getallen die echter na de decimale punt geen cijfers hebben, worden in dat geval aangevuld met een aantal nullen, gelijk aan het tweede getal in de vector. Als we nu bij de matrix M een formaat toepassen dat voor een uitvoer zorgt waar zes posities voor een getal genoeg zijn, waarvan twee gereserveerd voor de significante cijfers na het decimaal teken, dan schrijven we:

```

      6 2 ¶M
- 1.23  3.46
-14.00200.10

```

Alhoewel de getallen netjes in de kolom staan, hebben we tussen de kolommen zelf geen ruimte gelaten. Dit komt omdat de twee kolommen in de matrix elk zes posities krijgen toegewezen. In de tweede rij worden dus alle kolommen gebruikt wat tot het vreemde resultaat leidt. Het gebruik van de formaatvector 8 2, zal hiervoor beter zijn:

```

      8 2 ¶M
- 1.23    3.46
-14.00   200.10

```

Indien er echter niet genoeg plaats gereserveerd is voor de getallen, of het aantal significante cijfers voor de decimale punt is te groot, dan resulteert dit in een fout zoals in:

```

      5 2¶M
DOMAIN ERROR
      5 2¶M
      ^

```

Om deze laatste fout te vermijden kan men gebruik maken van een scalaire, echter alleen als het linkse gegeven van de formaatfunctie. Op die wijze wordt alleen aangegeven hoeveel significante cijfers na de decimale punt moeten worden afgedrukt. Voor de matrix M wordt dit:

```

      3  M
    1.230    3.456
- 14.000  200.100

```

Wanneer in de formaatvector, het linkse gegeven dus, een nul voorkomt, dan krijgen we een soortgelijke uitwerking. Is nul het eerste getal, dan is de uitwerking net als bij het gebruik van een scalaire. Is het tweede getal een nul (ook als alleenstaande scalaire) dan worden alle geformatteerde getallen naar gehele getallen afgerond.

In de monadische vorm wordt de formaat functie Φ gebruikt, net als de uitvoer-variantie $\square\leftarrow$, met andere woorden een numeriek gegeven wordt in een karaktervector omgezet. Als bijvoorbeeld de vector V een aantal getallen bevat worden deze, inclusief de spaties tussen de getallen, naar een karaktervector omgezet:

```

      V←6 14 1
      □←KV←ΦV
6 14 1
      ρKV
6
      KV×2
DOMAIN ERROR
      KV×2
      ^

```

Net als bij de andere uitvoervariantie, kan men verder geen numerieke bewerkingen meer uitvoeren met de karaktervector KV . Bij het koppelen van numerieke en alfanumerieke uitvoer zoals dat in het volgende deel van dit hoofdstuk besproken wordt, is het de bedoeling de functie in de monadische vorm te gebruiken.

Er zijn nog enkele *APL*-systemen die van een andere vorm van formattering gebruik maken. Het betreft hier een functie genaamd *DFT* die normaliter uit een voor iedereen toegankelijk werkgeheugen kan worden gecopieerd. De functie werkt op dezelfde manier als de functie Φ waar links één of twee getallen moeten staan. Deze functie werkt alleen maar wanneer de index-generator vanaf één begint, en het rechtse te formatteren gegeven niet groter is dan rangorde drie. Uiteraard kunnen ook hier slechts gehele getallen links van de functienaam worden gebruikt. Men moet bij het gebruik van de functie opletten dat, indien er twee getallen links gegeven zijn, het eerste getal altijd gelijk moet zijn aan het tweede getal +2, en dat indien er getallen zijn die met een te klein formaat moeten worden afgedrukt, de linkse significante cijfers worden weggelaten. Dit laatste staat in tegenstelling tot het gebruik van de functie Φ waarbij wel een foutmelding volgt.

Een paar voorbeelden van het gebruik van *DFT* volgen hieronder:

```
V←4.2 6.49 14.9 333.3333
6 1 DFT V
4.2 6.5 14.9 333.3
3 2 DFT V̇
DFT DOMAIN ERROR
```

Een laatste opmerking betreffende de functie gaat over het verbruik van *CVE-tijd* de tijd die nodig is voor de centrale verwerkingseenheid om de functie uit te voeren. Deze tijd is voor de meeste systemen groter dan voor het gebruik van de functie. De voorkeur gaat dus uit naar het gebruik van Φ .

7.5 Samenvoeging van numerieke en alfanumerieke gegevens

In deze paragraaf worden verschillende manieren besproken waarop alfanumerieke en numerieke gegevens aan elkaar kunnen worden gekoppeld. Reeds bij de uitvoer hebben we opgemerkt dat numerieke gegevens samen met tekst kunnen worden afgedrukt, zolang deze door een puntkomma van elkaar zijn gescheiden. In principe is het in *APL* niet toegestaan om twee types van vectoren aan elkaar te koppelen. Dit blijkt uit de volgende twee voorbeelden:

```
'HET GETAL IS ' ;X←?10
HET GETAL IS 4
'A'∈'HET GETAL IS ' ;X←?10
14
X
4
```

Indien we echter de waarde *X* toch door middel van een komma willen samenvoegen, dan moeten we eerst de numerieke waarde naar een karakter omzetten. Dit doen we door gebruik te maken van de functie Φ . Het resultaat van deze bewerking kan dan met het gebruik van de komma aan de tekst worden gekoppeld, zodat de vector *A* een alfanumerieke vector is met het getal in karakters. Het is dus correct te schrijven:

```
KV←'HET GETAL IS ' ,▼X
KV
HET GETAL IS 4
ρKV
14
'A'∈KV
1
```

Het is ook toegestaan de functie tussen twee vectoren met tekst te gebruiken, mits alleen het te converteren gedeelte wordt omgezet. We zetten dan het numerieke gedeelte tussen haakjes, anders resulteert het in een fout:

```
'DE AFSTAND IS ONGEVEER ', (100+120), ' KM'
DE AFSTAND IS ONGEVEER 106 KM
'DE AFSTAND IS ONGEVEER ', 100+120, ' KM'
DOMAIN ERROR
'DE AFSTAND IS ONGEVEER ', 100+120, ' KM'
      ^
```

Een andere manier om de gegevens aan elkaar te koppelen is gebruik te maken van het uitvoerteken $\boxplus$, dat meestal wordt gebruikt voor de invoer van alfanumerieke gegevens. Net als de functie $\boxplus$ kunnen we een getal omzetten in een karaktervector zoals in:

```
K $\boxplus$ 10
55
pK
```

Als uitgebreider voorbeeld schrijven we een functie die enigszins overeenkomt met elektronische spelletjes die tot doel hebben het leren optellen, aftrekken, vermenigvuldigen en delen van kleine en grote getallen. We tonen eerst de gedefinieerde functie om die dan te bespreken als toepassing van de omzetting van numerieke gegevens en het aan elkaar koppelen van gegevens van verschillende types. We noemen de functie *PLUS*:

```
 $\nabla$  PLUS
[1] I $\leftarrow$ 0
[2] NOG: $\rightarrow$ (10<I+I+1)/0
[3]  $\boxplus$ A $\leftarrow$ ?9
[4]  $\boxplus$  $\leftarrow$ ' + '
[5]  $\boxplus$ B $\leftarrow$ ?9
[6]  $\boxplus$  $\leftarrow$ ' = '
[7] ANTWOORD $\leftarrow$ 8 $\boxplus$  $\boxplus$ 
[8]  $\rightarrow$ ((pANTWOORD)=+ / ANTWOORD= $\boxplus$ A+B) / NOG
[9] 'FOUT ! ANTWOORD IS ',  $\boxplus$ A+B
[10]  $\rightarrow$ NOG
 $\nabla$ 
```

De regels [3] tot en met [7] zijn hier met name van belang. De numerieke waarden voor zowel *A* als voor *B* worden in karaktervorm geplaatst. De tekstgedeelten met + en = worden, zoals dat met de getallen gebeurt, in de juiste volgorde aan een uitvoervector $\boxplus$ gevoegd. Dit heeft als resultaat dat bij de uitvoer alles op één enkele regel komt te staan, inclusief het antwoord dat gegeven worden. De enige

voorwaarde is dat er geen *quad* teken □ tussen mag komen, anders wordt de uitvoervector verstoord en wordt het resterende gedeelte op een volgende regel afgedrukt. Het teken Φ wordt de executie-functie genoemd. Deze functie heeft als doel een alfanumerieke vector uit te voeren. In het voorgaande geval komt het neer op het omzetten van het antwoord in karaktersvorm naar zijn numeriek equivalent. Zodoende kan het op de volgende regel worden vergeleken met het berekende numerieke antwoord. Men moet er echter wel op letten dat de karaktervector op dat moment langer is dan het antwoord zelf. Omdat we hier altijd getallen van één cijfer kiezen, kunnen we als vaste regel zeven karakters weg laten vallen om alleen het antwoord zelf om te zetten. Een paar voorbeelden van de uitvoering van deze functie zijn:

```

                PLUS
      2 + 5 = 7
      9 + 8 = 18
      FOUR : ANTWOORD IS 17
      4 + 3 =  $\Phi$ 
      INTERRUPT
      PLUS[7] ANTWOORD←8+ $\Phi$ 
                ^
    
```

Variaties op deze functie zijn voor grotere getallen en het gebruik van andere functies heel eenvoudig te schrijven.

De executie-functie Φ die we in bovenstaand programma hebben gebruikt, past niet zozeer bij het onderwerp van samenvoeging van gegevens van verschillend type, maar past wel bij dit hoofdstuk omdat het bijna de omgekeerde functie heeft van de functie Φ .

De bedoeling van de functie is het evalueren of uitvoeren van de informatie die er rechts van is gegeven. Het gebruik is dus alleen monadisch. De eenvoudigste vorm is, zoals reeds getoond, het omzetten van getallen in karaktersvorm naar een numerieke vector, zoals in:

```

      N← $\Phi$ '4 6 7      9'
      N
      4 6 7 9
      ρN
      4
    
```

Ook in de executievorm is het gebruik eenvoudig, mits de informatie in *APL* een goede uitdrukking is. Bijvoorbeeld:

```

       $\Phi$  '4×6÷2'
      12
       $\Phi$  'A←10'
      A
    
```

```

      ^'5÷0'
DOMAIN ERROR
      5÷0
      ^
      ^')OFF'

```

Er kunnen heel wat toepassingen worden gevonden waar de executie-functie een belangrijke rol speelt. Zo kan men een gedefinieerde functie in matrixvorm schrijven en met behulp van Φ elke regel laten uitvoeren onder de voorwaarde dat alle regels moeten worden uitgevoerd en dat er niet wordt versprongen. Soms is het ook voordeliger om gegevens of bewerkingen in een karaktervector op te slaan omdat deze mogelijkerwijze in het werkgeheugen minder plaats innemen. De oefeningen aan het eind van dit hoofdstuk geven nog een paar mogelijkheden aan.

7.6 Wijziging van uitvoer in een gedefinieerde functie

Als voorbeeld van wijziging van de uitvoer, nemen we de functie *SCHETS* van hoofdstuk 6. Bij het tekenen hadden we aan de rechtse kant de getallen van de waarden in de vector *Y*. Deze waarden zijn echter een benadering omdat we op schaal tekenen en de waarden afronden. We hebben dus ook heel weinig aan een waarde .999943234 als .99 of afgerond één even goed is. De verschillende waarden staan ook nergens mooi in een kolom omdat er gehele en decimale getallen voorkomen. Daarbij komt nog dat de laatste uitgevoerde regel slechts twee getallen op de x-as aangeeft. Ook dit zullen we nu veranderen omdat we de mogelijkheid hebben op de juiste posities naast en onder de grafiek getallen te plaatsen.

Om de uitvoer van de y-waarden te veranderen, zullen we de getallen voor de getekende regel zelf afdrukken. Als we daarbij nog gebruik maken van een formaatvector die in de meeste gevallen groot genoeg zal zijn (zoals 8 2), dan veranderen we regel [16] als volgt:

```

      V$CHETS[16]
[16]  (8 2 V(Γ/Y)-W×I-1),'|';M[I;];'|'
[17]  V

```

De twee laatste regels in de grafische functie zijn:

```

      V$CHETS[20]
[20]  EINDE:' ' ;40p'----'|'
[21]  (L/X);(38p' ');Γ/X
[22]

```


De flexibiliteit van uitvoer van gegevens en resultaten is dus enorm vergroot. De meeste *APL*-functies zullen hier gebruik van maken, zoals blijkt uit de toepassingen in hoofdstuk 8.

7.7 Oefeningen

- Schrijf de correcte instructie voor de invoer van:
 - Twintig karakters, ongeacht de lengte van de ingetypte vector.
 - Tien getallen, ongeacht de lengte van de ingetypte vector.
 - Een getal dat naar een regel in een functie verwijst.
 - Een karakter dat, als het in een tekst voorkomt, aanleiding geeft tot het beëindigen van een functie.
- Geef het probleem aan met het hervatten van een regel in een functie, nadat deze op het punt $\wedge$ werd onderbroken:

```

□←A←Z÷Q←A
^
Q←Z FNCTX Q
^
→1 × I←I×5
^

```

- Wat is de uitvoer van de volgende instructies en geef aan waar een fout resulteert:

```

▽22 + ▽44
▽K←[ ]←+/110
2×Z←▽K←?10
2 2▽M←2 2ρ14
4 2▽M←2 2ρ14
2▽14×Q←2

```

- Wat is de uitvoer van de volgende instructies en geef aan waar een fout resulteert.

```

(▽I), ' IS ALTIJD ', ▽I
I; ' IS ALTIJD ', I
STOP: 'FUNCTIE STOPT OP REGEL ' ; ▽K←?10
[ ]←▽NM←2[ ]
6 2 ▽←[ /X←1 2 3 4 5

```

- Breng een verandering aan in de functie om te tekenen, zodat de gebruiker zelf het formaat kan aangeven om de waarden van variabele *Y* in de linkse kolom af te drukken.

8 *APL*-functies in de praktijk

Als vervolg op het derde hoofdstuk over programmeren in *APL* bestuderen we nu de verdere mogelijkheden bij het schrijven van een functie. Dit doen we aan de hand van enkele faciliteiten bij het programmeren, zoals het inbouwen van bepaalde beslissingsregels, het op verschillende manieren stoppen en voortzetten van een functie en enkele algemene wenken over het veranderen van de gedefinieerde functie.

Naast de reeds bekende functies ontwerpen we enkele functies die voor het dagelijks gebruik van pas kunnen komen. De geavanceerde functies in het laatste deel geven reeds een goede indruk van de rol die *APL* in de automatiseringswereld speelt.

8.1 Meer over gedefinieerde functies

8.1.1 Verzegelen van een functie

Een geschreven functie kan men op twee manieren behandelen. De functie blijft ofwel voor altijd geldig en men zal er dus geen veranderingen meer in aanbrengen, ofwel de functie moet regelmatig worden aangepast. Het eerste geval is normaal gezien de eigenlijke bedoeling van het programmeren. Een goede functie zal inderdaad zo weinig mogelijk moeten veranderen. Vele *APL*-programmeurs vinden bij het tweede geval dat de functie waardeloos is geworden en dat het beter is een geheel nieuwe functie te schrijven. In die gevallen waar geen veranderingen meer plaats zullen vinden, kan men een functie 'verzegelen' zodat niemand, dus ook de programmeur zelf, er verder nog in kan kijken of veranderingen aanbrengen. Om een functie te kunnen verzegelen moet men de functie eerst openen. Normaliter sluit men de functie af met het teken ∇ , maar om te verzegelen sluit men deze af met het teken ∇ , of een samenstelling van de tekens ∇ en $\sim$. De functie kan dan niet meer worden geopend om te kijken wat er op de verschillende regels staat. Men kan alleen nog maar de functie uitvoeren of uit het werkgeheugen verwijderen. Het is dus altijd aan te raden een kopie van de functie op papier of in een andere werkruimte te bewaren. Als voorbeeld tonen we een functie met de naam *F* die we nog open laten en daarna sluiten en verzegelen om dan weer op te roepen:

```

      ∇F[□]
[1]   □←K←?10
[2]   →1×10≠K
      ∇
[3]   ↵

      F

4
6
7
10

```

```

      ∇F
DEFN ERROR
      ∇F
      ^
      ∇F[2□]
DEFN ERROR
      ^

```

Soms kan een systeemprogrammeur de *APL* functie weer openmaken. Hier kan men beter niet op rekenen, want dit is geen vaste regel.

8.1.2 Verwijderen van regels binnen een functie

Bij het openen van de functie hebben we gebruik gemaakt van het *quad* teken □. Als daar geen regelnummer bij vermeld staat, wordt de gehele functie getoond. Wil men echter de functie vanaf een bepaald regelnummer laten zien, dan geeft men dat nummer na het teken □ op. Om een bepaalde regel alleen te zien, zet men het regelnummer vóór het teken □. In alle gevallen kan men na het sluiten van het vierkante haakje een sluitteken ∇ meegeven, zodat na het tonen van één regel, een aantal regels, of de gehele functie, de functie meteen weer wordt gesloten.

Voor het verwijderen van regels in een functie hebben we gezien dat het opvragen van de regel met het regelnummer tussen vierkante haakjes volstaat als dat wordt gevolgd door een druk op de ATTENTION-toets en de RETURN-toets. Bij verschillende *APL*-systemen werkt deze procedure anders waar ze is vervangen door een meer bewuste handeling zodat niet per ongeluk de verkeerde regels worden verwijderd.

De instructie hiervoor lijkt op het laten zien van een bepaalde regel, bijvoorbeeld regel zeven, door [□7] in te voeren, nadat de functie geopend is. Men vervangt nu het *quad* teken □ door het driehoekje △, dus [△7]. Bijvoorbeeld:


```

      ∇  LOSΔOP X
[1]    X ← ∇
[2]    'STAAT DE LETTER ', X, ' IN DE TEKST?'
[3]    A ← 'DIT IS DE GEHEIME TEKST'
[4]    → (X ∈ A) / JA
[5]    'NEEN'
[6]    → 0
[7]    JA: 'JA'
      ∇

```

We kunnen in de bovenstaande functie regel [1] laten wegvallen zodra de functie is geopend. We doen dit omdat de variabele X reeds bij de definitie van de functie is gegeven. Een eventuele vraag naar de eerste regel levert een blanco regel op, waarna men de functie kan sluiten. De functie telt nu één regel minder zodat het laatste regelnummer 7 is geworden.

8.1.3 Variaties op verwijzingen binnen een functie

Het verdere verloop van de bovenstaande functie staat nu vast. Op de nieuwe regel [3] kunnen we desgewenst een variatie van de verwijzing inbrengen. Het resultaat van $A \in X$ is immers nul of één, afhankelijk van het feit of de gekozen letter niet of wel in de tekst voorkomt. Als we het getal één bij dit resultaat optellen, krijgen we als nieuw resultaat één of twee, we als index gebruiken op dezelfde regel:

```
[3] → 4 6[1+X ∈ A]
```

De waarden vier en zes vormen een vector die geïndiceerd werd met het resultaat van de test vermeerderd met één. Komt de letter in de tekst voor, dan is de index twee en er wordt naar regel zes gesprongen waar het positieve antwoord gegeven wordt. In het andere geval wordt naar regel vier gesprongen.

Een variatie hierop is bijvoorbeeld een veelvoudige verwijzing, wanneer verschillende mogelijkheden worden geboden. Stel dat in een functie de variabele I is ingevoerd, en dat er afhankelijk van de waarde van deze variabele een verwijzing plaatsvindt naar een specifieke regel. Dit doen we als volgt:

```
→ 10 20 30 40 50 [(15)1I]
```

Indien de waarde van variabele I groter is dan vijf of kleiner dan één, krijgt men geen indexfout, maar volgt er een verwijzing naar regel nul alsof er stond $\rightarrow 0$, wat overeenkomt met een beëindiging van de functie.

8.2 Het onderbreken en hervatten van een functie

Bij het schrijven van *APL*-functies maakt men wel eens fouten met betrekking tot de regels van de syntax. Er komt dan een foutmelding tijdens het uitvoeren van de functie. Als men de regel kan herstellen dan begint men over het algemeen opnieuw met het uitvoeren. Het *APL*-systeem houdt echter bij waar de functies zijn gestopt of onderbroken. Onder bepaalde omstandigheden kan de functie vanaf de gestopte plaats worden voortgezet.

Er zijn een zestal oorzaken waarom een functie zou kunnen stoppen. Enkele hiervan zijn we bij de oefeningen al tegengekomen omdat het de meest voorkomende en meest voor de hand liggende onderbrekingen zijn. De zes oorzaken zijn:

1. Een fout in een regel.
2. Een ingrijpen van de computer of zijn beheerder.
3. De beheerder stuurde een algemeen bericht.
4. Bij de vraag naar invoer is *OUT* getypt.
5. De *ATTENTION*-toets is ingedrukt.
6. Een vector met *STOP*-controles is gedefinieerd.

We bespreken afzonderlijk de eerste drie oorzaken omdat deze met een vorm van ongewenste onderbreking overeenkomen en dan de laatste drie omdat de *APL*-gebruiker daar enige vorm van controle heeft over het stoppen van de aan de gang zijnde functie.

8.2.1 Ongewenste onderbrekingen

Stel dat een functie als volgt geschreven is:

```

      ∇TEST
[1]  I←?10
[2]  →(I>3)/0
[3]  MACHT
[4]  →1
      ∇

```

Bij het uitvoeren van deze functie blijkt nu dat op regelnummer drie de waarde van de variabele *MACHT* niet bekend is. Het systeem heeft op dat moment een notitie gemaakt van de status van de functie. Dit kan men nakijken door het invoeren van het commando *)SI*, waarop het *APL*-systeem antwoordt met een lijst met de status van de onderbroken functies. We noemen deze lijst de *statuslijst*. Voor het voorbeeld krijgt men dan het volgende bericht:

```

      )SI
TEST[3] *

```


De bedoeling was geweest een andere functie te schrijven met de naam *MACHT*. Omdat deze functie nog niet was gedefinieerd heeft *APL* in eerste instantie nagekeken of er een variabele met die naam in het werkgeheugen voorkwam. Dit was niet het geval. De status die de plaats van onderbreking in de functie aangeeft kunnen we nu weghalen door het aanwijzingsteken $\rightarrow$ niladisch te gebruiken. De statuslijst is dan leeg. Voorbeeld:

```
→
)SI
```

Na het invoeren van *)SI* is de lijst inderdaad leeg.

Bij herhaald gebruik van een foutieve functie komt de vermelding van deze functie steeds vaker voor. Dit gebeurt in de volgorde van het maken van fouten, ook als andere functies onderbroken werden. Een paar voorbeelden geven aan hoe dit werkt:

```

      VZES
[1]   I←1
[2]   →(6<I←I+1)/0
[3]   NEEM I
[4]   →2V

      )SI
      TEST
SYNTAX ERROR
TEST[3] MACHT
      ^
      )SI
TEST[3] *
      ZES
SYNTAX ERROR
ZES[3] NEEM I
      ^
      )SI
ZES[3] *
TEST[3] *
```

De status van de onderbroken functies is in die volgorde bijgehouden, waar de jongste onderbreking bovenaan staat en de oudste onderbreking onderaan in de lijst. Bij het niladisch gebruik van het teken $\rightarrow$ wordt altijd de bovenste status, dus die van de laatst onderbroken functie, verwijderd. We krijgen dan:

```
→
)SI
TEST[3] *
```

De functie *TEST* staat nu als jongst onderbroken functie genoteerd. Willen we nog eens proberen om die functie uit te voeren dan laten we het verwijzingspijlje $\rightarrow$ volgen door het regelnummer dat bij de status van de functie vermeld staat. Dit zou dus $\rightarrow 3$ moeten zijn. Omdat de fout op deze regel nog niet hersteld is komt de status van de functie opnieuw als jongste in de statuslijst.

We verwijderen alle referenties naar onderbroken functies en starten de functie *ZES* opnieuw:

```

→
)SI
ZES
SYNTAX ERROR
ZES[3] NEEM I
    ^
    )SI
ZES[3] *
```

Als men nu besluit de functienaam zelf te wijzigen of regels aan de functie toe te voegen, dan gebeurt er met de status voor de gewijzigde en onderbroken functie het volgende:

```

      ∇ZES[1.1]
[1.1] A
SI DAMAGE
[1.2] ∇
      )SI
ZES[*]
```

De status is vernietigd omdat bij het sluiten van de functie de oude regel [2] regelnummer [3] zal zijn geworden. Zelfs als er geen regel tussengevoegd is maar een andere regel in de functie veranderd, dan nog is de vermelding van de status van de functie in de lijst weggevallen. Een alleenstaande asterisk in de lijst geeft aan dat er een functie onderbroken was en niet meer met een verwijzing kan worden hervat. Het is goed regelmatig deze lijst na te kijken en een ongewenste status met de niladische verwijzing $\rightarrow$ te verwijderen.

De tweede en derde oorzaak waarom een functie zou kunnen worden onderbroken liggen niet noodzakelijkerwijze bij de programmeur. Bij grotere computersystemen (dus niet bij microcomputers) gebeurt het regelmatig dat er een algemeen bericht rondgestuurd wordt door de beheerder van het *APL*-systeem, dit in verband met sluitingen, onderhoud, enz. Het vervelende is dan dat als er net een functie wordt uitgevoerd, deze ook onderbroken wordt. Men kan dan in de statuslijst kijken waar de functie gestopt werd en daarna door het intypen van $\rightarrow$ en het regelnummer, de functie weer hervatten. Men moet echter opletten dat de

functie niet in het midden van een bepaalde regel onderbroken werd. Bij de onderbreking zelf kan men dit zien zoals bijvoorbeeld:

```
PA! MORGENMIDDAG VAN 3 TOT 5 UUR ONDERHOUD.
INTERRUPT
SCHETS[5] RIJ+RIJ+0=RIJ+RIJ+(20<RIJ)*-1
          ^
        )SI
SCHETS[5] *
ZES[*]
```

Als we meteen $\rightarrow 5$ invoeren dan moeten we rekening houden met het feit dat de regel weer volledig van rechts naar links wordt uitgevoerd. Hierbij kan de variabele *RIJ* een vector zijn met andere waarden dan werd verwacht, omdat de functie hervat wordt met de waarden die tot op het moment van de onderbreking berekend waren. Weet men zeker dat er geen problemen zijn, dan kan men de functie rustig hervatten.

Het is ook mogelijk dat de beheerder de gebruiker uit het systeem 'gooit' zodat lopende functies worden onderbroken. De computer zelf kan dit ook doen, door bijvoorbeeld bij een veel te hoog gebruik van rekentijd de bewerkingen te laten stoppen. Ook in dat geval kan men de functie op de hierboven beschreven manier voortzetten.

We keren terug naar de functie *TEST* waar we op regel [3] een andere functie willen oproepen. We definiëren deze functie als volgt:

```
VMACHT
[1] (V I), ' TOT DE MACHT ', (V I), ' IS ', I*I V
```

De meest rechtse *komma* op de enige regel moet een *punt komma* zijn omdat anders een alfanumerieke en een numerieke vector zonder omzetting aan elkaar worden gevoegd. Executie van de functie *TEST* leidt nu tot de volgende fout en status:

```
TEST
DOMAIN ERROR
MACHT[1] (V), ' TOT DE MACHT ', (V I), ' IS ', I*I
          ^
        )SI
MACHT[1] *
TEST[3]
```

De statuslijst geeft aan dat de fout in de 'hulpfunctie' zit, en dat deze hulpfunctie in *TEST* op regel [3] werd opgeroepen. We veranderen de foutieve regel, wat niet tot een *SI DAMAGE* leidt! We verwijderen de status in de lijst niet, maar

hervatten de functie met $\rightarrow I$. De functies eindigen normaal en de statuslijst bevat geen referenties naar onderbroken functies meer zoals blijkt uit het onderstaande vervolg van de functies:

```

VMACHT[1□35]
[1] (▽I),' TOT DE MACHT ',(▽I),' IS ',I*I
      /1
[1] (▽I),' TOT DE MACHT ',(▽I),' IS ' ;I*I
[2] ▽
      →1
3 TOT DE MACHT 3 IS 27
2 TOT DE MACHT 2 IS 4
)SI

```

8.2.2 Gewenste onderbrekingen

Het gebruik van de ATTENTION-toets, reeds in hoofdstuk 6 besproken, heeft tot gevolg dat de functie tijdens het uitvoeren wordt gestopt. De plaats van onderbreking wordt door het systeem afgedrukt en de functie kan op de normale manier en met de nodige voorzorgen worden hervat. Als voorbeeld voor het gebruik van de ATTENTION-toets, stoppen we een functie:

```

10 ΔENΔ 10
VERDELING IN BRIEFJES EN LOSSE GULDENS
0 KEER 100
0 KEER 25
1 KEER 10
INTERRUPT
VERDEELΔIN[7] AANTAL;' KEER ' ;X[I]
      ^
)SI
VERDEELΔIN[7] *
ΔENΔ[5]

```

Zoals trouwens voor alle onderbrekingen geldt, mag de functie met de uitvoering van een willekeurige regel voortgezet worden. Men hoeft eigenlijk nooit naar de onderbroken regel zelf te verwijzen. Een verwijzing naar regel [1] is ook toegestaan.

Bij het stoppen van de functie tijdens een vraag naar invoer, krijgt men opnieuw dezelfde situatie als bij de functie *DATA* van hoofdstuk 2:


```

DATA 5
□:
  →
  INTERRUPT
  DATA[2] A←,□
                ^

```

Zolang het invoerteken □ (of □) op de meest rechtse plaats in de regel staat is er geen probleem met voortzetting. Indien dit niet het geval is moet men eerst nagaan of er voor een tweede keer bewerkingen plaatsvinden, hetgeen in sommige gevallen niet wenselijk is.

De laatste manier waarop men een functie kan stoppen is door het definiëren van een STOP-vector. Dit doet men op de volgende wijze:

*S*Δ*TEST*←2

De letter *S* gevolgd door het rechtopstaande driehoekje Δ betekent dat voor de functienaam die volgt de stopvector wordt bepaald. De functienaam moet een naam zijn van een functie die in het werkgeheugen voorkomt. De toewijzing met het teken ← geeft dan aan op welke regel of regels de functie moet worden gestopt. Het stoppen zelf gebeurt net vóór het uitvoeren van de bewerkingen op die regel(s). Voorbeeld:

```

TEST
TEST[2]
  I
3
  →2
3 TOT DE MACHT 3 IS 27
TEST[2]
  I
7
  →2

```

De stopvector is geen variabele en men kan dus ook niet opvragen op welke regels er moet worden gestopt. Bij het verwijderen van de functie uit het werkgeheugen of bij het toevoegen of invoegen van regels in de functie wordt, althans bij de meeste *APL*-systemen, de stopvector ook verwijderd.

Indien er geen stopvector voor een functie meer nodig is, kan men deze op nul stellen door in te voeren:

*S*Δ*TEST*←10

De stopvector kan ook in de functie zelf worden bepaald, waarbij de regelnummers waarop moet worden gestopt eventueel door een formule kunnen worden berekend. Uiteraard kan men een regelnummer door een regelnaam substitueren, zoals uit onderstaand voorbeeld blijkt:

```

      VHALT
[1]  SΔHALT←3,HIER
[2]  X←110
[3]  Y←10X÷3
[4]  HIER:X SCHETS Y
[5]  'EINDE SCHETS'▽
      HALT
HALT[3]
      X
1 2 3 4 5 6 7 8 9 10
      →3
HALT[4]
      )SI
HALT[4] *
```

8.2.3 Resultaten per regel

Heel dikwijls wordt de stopvector gebruikt om op een bepaalde regel in de functie te zien wat de waarden van de verschillende berekende variabelen zijn. Ook de lokale variabelen kunnen dan worden bekeken omdat de functie in een onderbroken toestand verkeert. Als de functie is beëindigd, kunnen alleen nog de globale variabelen worden bekeken. In APL is ook een faciliteit ingebouwd die wat betreft de opbouw met de stopvector overeenkomt. Het is namelijk mogelijk de resultaten van regels in een functie te laten zien tijdens het uitvoeren van de functie. In het Engels heet dat *trace*. De *trace*-vector wordt gevormd door de letter *T* gevolgd door het teken Δ , de functienaam, de toewijzing $\leftarrow$ en de regel(s) met het resultaat van een instructie (als er een resultaat is). In de praktijk werkt dit als volgt:

```

      TΔTEST←13
      TEST
      →
INTERRUPT
DATA[2] A←,□
      ^
TEST[1] 2
TEST[2]
2 TOT DE MACHT 2 IS 4
TEST[3]
TEST[1] 4
TEST[2] 0
```


Op regel [3] wordt niets berekend en er is dus geen resultaat. Op de regel waar een verwijzing plaatsvindt, wordt als resultaat niet de waarde van de toets weergegeven, maar wel het regelnummer waarnaar verwezen wordt. We stellen de *trace*-vector voor de functie *TEST* op nul, en gaan nu kijken hoe de tussentijdse resultaten van de functie *SCHETS* verlopen. Om de tekening zelf te vermijden, bepalen we de stopvector voor regel [9]:

```

TΔTEST←1 0
TΔSCHETS←2 5 6 8
SΔSCHETS←9
X←Y←1 10
X SCHETS Y
SCHETS[2] 2
SCHETS[5] 2 4 6 8 10 12 14 16 18 20
SCHETS[6] 4
SCHETS[8] 4 8 12 16 20 24 28 32 36 40
SCHETS[9]

```

8.3 Enkele eenvoudige hulpfuncties

In de praktijk is het soms wenselijk over enkele functies te beschikken die als onderdeel kunnen worden beschouwd van een andere functie omdat ze bijvoorbeeld altijd terugkerende bewerkingen bevatten. Bepaalde primitieve functies in de *APL* taal vervullen deze opdracht reeds. Als we bijvoorbeeld kijken naar de berekening van de inverse van een matrix ($\boxplus$), de sinuswaarde (IO), decoding ($\top$), enz. Indien gedefinieerde functies een 'onderdeel' vormen kunnen we spreken van hulpfuncties.

De volgende hulpfuncties kunnen ook als afzonderlijke gedefinieerde functies worden beschouwd en als zodanig kan men ze ook uitvoeren. Enkele zijn reeds bij oefeningen van pas gekomen, terwijl er een paar volledig nieuw zijn.

8.3.1 Meervoudige instructies op een regel

Bij de toewijzing van variabelen hebben we reeds gebruik gemaakt van het in één enkele keer op dezelfde regel toekennen van waarden aan verschillende variabelen. Dit doen we bijvoorbeeld als volgt:

```
X←Y←1 10
```

De mogelijkheid bestaat om volgens de regels van de *APL*-syntax, verschillende instructies op een regel te plaatsen. Daartoe maken we bijvoorbeeld gebruik van

de samenvoeging van instructies door middel van de 'nul-dimensie' ,0p, zoals in:

$$M \leftarrow (5,4) \rho 0,0pI \leftarrow 15,0pV \leftarrow Y \leftarrow 4 \quad 8 \quad 10 \quad 14 \quad 17$$

$$M[I;1] \leftarrow V[I],0pM[I;2] \leftarrow Y[1]$$

Dit gebruik is misschien wel handig, maar het leidt over het algemeen tot verwarring voor anderen die de functies later moeten lezen. Het gebruik van deze vorm raden we af. De meeste leveranciers van APL-systemen hebben zelf een teken gekozen dat toelaat verschillende instructies op een regel te schrijven. Dit wordt dan meestal het $\diamond$ teken dat op een regel tussen twee instructies mag worden geschreven. Omdat dit teken sterk afhangt van het APL-systeem, wordt het hier niet verder besproken. Hetzelfde resultaat is in meer algemene zin te verkrijgen.

Het is namelijk heel eenvoudig zelf een functie te schrijven die door een unieke naam gemakkelijk herkenbaar is. De functie heeft twee gegevens bij executie en wordt dus in de dyadische vorm gebruikt. Het rechtse gegeven is de eerst uit te voeren instructie, en de nog uit te voeren instructies vormen het linkse gegeven. Als functienaam kiezen we $\triangle$ omdat het teken $\triangle$ zelf geen primitieve functie of functie is. De functie ziet er dan als volgt uit:

$$\nabla Z \leftarrow \text{LINKS } \triangle \text{ RECHTS}$$

$$[1] \quad Z \leftarrow \text{LINKS} \nabla$$

De instructie is zeer eenvoudig. Het gedeelte rechts wordt normaal uitgevoerd. Het gedeelte links vormt de uitvoer van de hulpfunctie en zal in de hoofdfunctie worden uitgevoerd. Een toepassing hiervan is:

$$\nabla KORT$$

$$[1] \quad \square \leftarrow I \leftarrow 1 \quad \triangle \quad \square \leftarrow J \leftarrow 2 \quad \triangle \quad \square \leftarrow K \leftarrow 3$$

$$[2] \quad \square \leftarrow I \times J \times K \quad \triangle \quad \nabla \leftarrow 'HET \text{ PRODUKT VAN } I, J \text{ EN } K \text{ IS } '$$

$$[3] \quad \nabla$$

$$KORT$$

$$3$$

$$2$$

$$1$$

$$HET \text{ PRODUKT VAN } I, J \text{ EN } K \text{ IS } 6$$

Uiteraard mag men voor deze hulpfunctie iedere naam kiezen, zolang de naam eenvoudig en makkelijk te herkennen is. Het is aan te raden deze functie alleen maar te gebruiken op die regels waar instructies staan die bij eventuele onderbreking opnieuw mogen worden uitgevoerd. Deze functie is handig om aan elkaar verwante berekeningen te groeperen. Als een test met verwijzing op de regel

voorkomt, kan men de functie $\triangle$ beter niet gebruiken. Dit laatste moet voor de beginnende APL-programmeur een vaste regel zijn.

8.3.2 Het testen van condities

Bij het testen van bepaalde waarden of berekeningen om te zien of ze aan één of andere voorwaarde voldoen kan men de verschillende methoden van verwijzing gebruiken. We hebben hier echter als voorbeeld een aparte functie geschreven die het toelaat de vraagstelling in het Nederlands te stellen, en we noemen de functie *ALS*. Deze functie is niet meer zo eenvoudig te schrijven als de voorgaande, omdat we hier met het feit rekening moeten houden dat enkele fouten in de test zelf kunnen worden gemaakt door bijvoorbeeld primitieve functies of geschreven functies verkeerd te gebruiken. Om er zeker van te zijn dat er geen fouten met betrekking tot de dimensies en de lengte voorkomen, maken we gebruik van nog een andere hulpfunctie die we *FOUT* noemen.

In de functie *ALS* gaan we eerst na of er in het voorwaarde gedeelte geen dimensiefout is voorgekomen. Dat doen we door de instructie:

```
' DIMENSIE VOORWAARDE ' FOUT 1≠ρ,VOORWAARDE
```

De functie *FOUT* is dyadisch. Het linkse gedeelte is de tekst die de fout beschrijft. Het gedeelte rechts beschrijft de voorwaarde waaronder de dimensie juist is. De functie definiëren en programmeren we als volgt:

```
VMELDING FOUT VOORWAARDE
[1] →(1≠v/,VOORWAARDE×VOORWAARDE=1)/0
[2] 'FOUT IN ',MELDING ∇
[3] →
```

Als we op de eerste regel geen één krijgen betekent dit dat aan de voorwaarde is voldaan en dat we de functie kunnen verlaten. In het andere geval wordt een fout gemeld. Let wel, op regelnummer drie. Als er een fout gemeld is wordt naar regel nul verwezen wat betekent dat de functie stopt! De functie *ALS* stopt dan ook omdat de fout ernstig is. Zonder het regelnummer drie zou de *ALS* functie verder gaan, alsook de functie die deze laatste gebruikt heeft.

De rest van de functie *ALS* ziet er als volgt uit:

```
VREGELNAAM ALS VOORWAARDE
[1] 'DIMENSIE VOORWAARDE ' FOUT 1≠ρ,VOORWAARDE
[2] 'REGELNAAM ' FOUT 1≠ρ,REGELNAAM
[3] 'DOMEIN ' FOUT ~VOORWAARDE∈ 0 1
[4] Z←VOORWAARDE/REGELNAAM ∇
```

Enkele toepassingen zijn:

```

      VTOETS
[1]  I←□
[2]  →LIJN4  ALS  I=3
[3]  →1  ALS  I≠3
[4]  LIJN4: 'GOED ZO'
[5]  ∇

```

```

      TOETS
□:
      2
□:
      3
GOED ZO

```

De hulpfunctie *FOUT* kan ook nog afzonderlijk worden gebruikt zonder de functie *ALS* hierin te betrekken. Een voorbeeld:

```

      'LENGTE INVOER (=10)' FOUT 10≠ρ,□
□:
      1 2 3 4 5 6 7 8 9
FOUT IN LENGTE INVOER (=10)

      'TEKEN GETAL (+)' FOUT 0>□
□:
      -2
FOUT IN TEKEN GETAL (+)

```

8.3.3 Het tekenen van een histogram

Het tekenen van een histogram kan soms heel nuttig zijn om het verloop van uitgaven, uitslagen, procentuele verdelingen en dergelijke op een visuele manier weer te geven. Als hulpfunctie gebruiken we een gedefinieerde functie die het histogram tekent. Men kan de functie *SCHETS* nemen en er een paar wijzigingen in aanbrengen, bijvoorbeeld in het opvullen van de kolommen waar iets getekend is.

Bij het berekenen van de vorm van het histogram gaan we van een paar vaste gegevens uit. Een variabele wordt gedefinieerd als een vector van lengte twee. Het eerste element is in aantal lijnen de hoogte van het histogram en het tweede element is het aantal spaties dat tussen de kolommen dient te staan. We bepalen deze vector als het linkse gegeven van de functie. Het rechtse gegeven wordt gevormd door de variabele die de te tekenen waarden bevat. De histogram functie

is dus een dyadische functie. Alle waarden in het rechtse gegeven moeten positief zijn, zoniet dan laten we functie stoppen. De definitie van de functie is:

∇ HS HISTOGRAM Y

De variabele HS is de vector met de waarde voor hoogte en spaties, terwijl de variabele Y de te tekenen waarden weergeven. Hieronder volgt dan de totale functie. Let hierbij op de schaalberekening en de afronding. Onder de kolommen wordt een volgnummer geschreven:

```

[1]  $\nabla$  HS HISTOGRAM Y;SCHAALY;W;RIJ;M;I;J
[2]  $\rightarrow((\lfloor Y \rfloor < 0) / \text{NEGATIEF}$ 
[3]  $\rightarrow \text{TWEE ALS } 2 \neq \rho, \text{HS}$ 
[4]  $\text{SCHAALY} \leftarrow 1 \div W \leftarrow (1 \div \text{HS}[1]) \times (1 \times 0 \neq \lfloor Y \rfloor) + (\lceil Y \rceil - \lfloor Y \rfloor)$ 
[5]  $\text{RIJ} \leftarrow \lfloor Y \rfloor \times \text{SCHAALY}$ 
[6]  $\text{RIJ} \leftarrow \text{RIJ} + 0 = \text{RIJ} \leftarrow \text{RIJ} + (\text{HS}[1] < \text{RIJ}) \times -1$ 
[7]  $M \leftarrow ((\text{HS}[2] \times \rho Y), \text{HS}[1]) \rho ' ' \Delta \square \leftarrow 2 \ 1 \ \rho ' '$ 
[8]  $I \leftarrow 0 \ \Delta \text{KOLOM} \leftarrow 1 - \text{HS}[2]$ 
[9]  $\text{TERUG1} \rightarrow ((\rho Y) < I \leftarrow I + 1) / \text{TEKEN}$ 
[10]  $J \leftarrow 0 \ \Delta \text{KOLOM} \leftarrow \text{KOLOM} + \text{HS}[2]$ 
[11]  $\text{TERUG2} \rightarrow (\text{RIJ}[I] < J \leftarrow J + 1) / \text{TERUG1}$ 
[12]  $M[\text{KOLOM}; J] \leftarrow '*'$ 
[13]  $\rightarrow \text{TERUG2}$ 
[14]  $\text{TEKEN} : I \leftarrow 0 \ \Delta M \leftarrow \ominus M \ \Delta M \leftarrow \ominus M$ 
[15]  $\text{PER} \Delta \text{RIJ} \rightarrow (\overline{\text{HS}}[1] < I \leftarrow I + 1) / \text{EINDE}$ 
[16]  $'|'; M[I;]; (\lceil Y \rceil - W \times I - 1$ 
[17]  $\rightarrow \text{PER} \Delta \text{RIJ}$ 
[18]  $\text{EINDE} : ' ' , (-1 \div \rho M) \rho ' -'$ 
[19]  $(\text{HS}[2] - 2) \div (\text{HS}[2], 0) \nabla \rho Y$ 
[20]  $\rightarrow 0$ 
[21]  $\text{NEGATIEF} : \text{'GEEN NEGATIEVE GETALLEN TOEGESTAAN - OPNIEUW'}$ 
[22]  $\rightarrow 0$ 
[23]  $\text{TWEE} : \text{'HET LINKER ARGUMENT VAN DE FUNCTIE MOET TWEE'}$ 
[24]  $\text{'GETALLEN BEVATTEN; 1 VOOR HOOGTE EN 1 VOOR AFSTAND'}$ 
[25]  $\text{'TUSSEN DE KOLOMMEN'}$ 

```

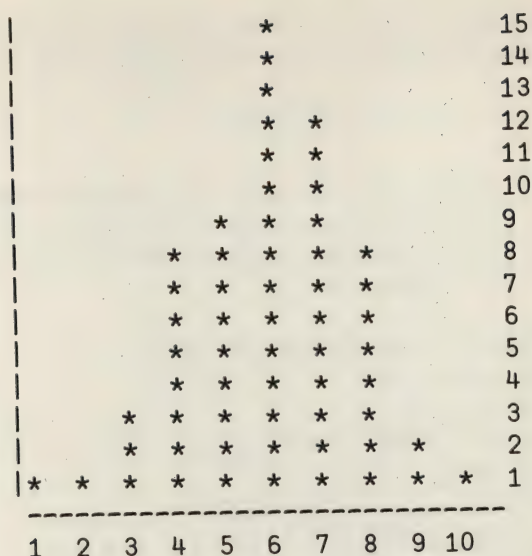
∇

Als praktisch voorbeeld nemen we een verdeling van de behaalde punten bij een tentamen. Waar geen enkele student die score gehaald heeft, brengen we een nul in. Het geheel wordt dan:

```

EX  $\leftarrow$  0 1 3 8 9 15 12 8 2 0
15 3 HISTOGRAM EX

```



Waar mogelijk maken we verder in dit hoofdstuk gebruik van deze hulpfuncties. Voor de beginnende programmeur in *APL* is het goed enkele van deze functies over te nemen, ook al begrijpt men in het begin niet goed hoe bepaalde dingen precies werken. Echter, oefening baart kunst! Wijzigingen die men er na verloop van tijd zelf inbrengt zullen de functies hopelijk nog verbeteren.

8.4 *APL*-toepassingen voor de praktijk

Niet alleen hulpfuncties vormen een belangrijk deel van een werkgeheugen, maar ook functies die dagelijks als spel of als serieuze toepassing kunnen worden gebruikt. Afhankelijk van de interesse van de *APL*-programmeur zullen er functies worden ontworpen die regelmatig moeten worden gebruikt. Soms moet men dagelijks nieuwe gegevens invoeren. Bepaalde functies zullen bijvoorbeeld regelmatig bedrijfsresultaten en -verwachtingen moeten geven.

Als voorbeelden van een toepassing beginnen we met een eenvoudige vorm van tekstverwerking, namelijk het invoeren van tekst, het opzoeken van woorden in die tekst, het wijzigen ervan, en het zodanig laten afdrucken dat de tekst zowel links als rechts een rechte kantlijn heeft. Vervolgens bespreken we een voorbeeld van huishoudelijke budgettering. Het voorbeeld van statistiek is niet moeilijk en kan naar wens worden uitgebreid. Voor het onderwerp computer ondersteund onderwijs (COO) is een voorbeeld gekozen dat aansluit op de functie *PLUS* uit hoofdstuk 7. Een laatste voorbeeld van financiële analyse geeft een goede indruk van de mogelijkheden van het werken met vectoren en matrices.

8.4.1 Eenvoudige tekstverwerking

In de automatiseringswereld van vandaag kunnen we vooral op kantoor het begrip tekstverwerking niet meer ontlopen. Voor bijna alle microcomputers is het mogelijk een programma te kopen waarmee men tekst kan verwerken. De functie die we hier programmeren is een vereenvoudigde vorm van tekstverwerking. Het spreekt vanzelf dat de *APL*-karakters niet geschikt zijn voor correspondentie. Soms is het wel mogelijk op de terminal een ander karakterttype aan te brengen, zodat de uitvoer van de *APL*-functie wel geschikt is voor kantoorgebruik. Aangezien dit geheel afhangt van de omstandigheden, raden we de lezer aan de mogelijkheden ter plaatse na te gaan.

In de te schrijven functie laten we vijf mogelijkheden toe:

1. Het invoeren van nieuwe tekst.
2. Het bijvoegen van tekst na de reeds ingevoerde tekst.
3. Het veranderen van korte gedeelten in de tekst.
4. Het afdrukken van de tekst.
5. Het stoppen van de tekstverwerking.

We kiezen voor de tekst zelf een vector met de naam *TEKST* als globale variabele. De functie noemen we *TV* en is op een gebruikersvriendelijke manier geschreven. Hiervoor maken we gebruik van een paar hulpfuncties die indien nodig, eventueel afzonderlijk kunnen worden gebruikt, maar die toch vanuit de hoofdfunctie *TV* worden opgeroepen.

Voor de gebruiker van de tekstverwerkende functie laten we eerst de mogelijkheden zien waaruit dan een keuze kan worden gemaakt. Dit programmeren we als volgt:

```

      ∇ TV;KEUZE;I;T
[1]  KEUZE←'1. INBRENGEN NIEUWE TEKST'
[2]  KEUZE←KEUZE,'2. BIJVOEGEN NIEUWE TEKST'
[3]  KEUZE←KEUZE,'3. WIJZIGEN VAN TEKST'
[4]  KEUZE←KEUZE,'4. AFDRUKKEN VAN TEKST'
[5]  KEUZE←KEUZE,'5. STOP'
[6]  KIES:'KIES UIT DE VOLGENDE MOGELIJKHEDEN:'
[7]  I←⌈▽⌋← 5 25 ρKEUZE

```

Op de laatste regel wordt de keuze van de gebruiker geëvalueerd en wordt meteen naar het gedeelte in de functie overgegaan waar de gekozen mogelijkheid moet worden uitgevoerd. Hierbij is door de programmeur besloten dat de invoer van nieuwe tekst en het bijvoegen van tekst in de hoofdfunctie zelf mag gebeuren. De mogelijkheden tot het veranderen van delen van de tekst of het afdrukken van de tekst zijn door twee afzonderlijke hulpfuncties te verwezenlijken. De functie *TV* ziet er dan in zijn geheel uit als:

```

[8]      →((I=1),(I=2),(I=3),(I=4),(5=I))/IN1,IN2,WIJZ,AF,STOP
[9]      IN1:'TYPE TEKST IN, OM TE STOPPEN TYPE *STOP* BIJ HET'
[10]     TEKST←10 Δ□ ←'BEGIN VAN EEN NIEUWE REGEL'
[11]     INX:→(6=+/(6†T+□,' ')= '*STOP*')/KIES
[12]     →INX Δ TEKST←TEKST,T
[13]     IN2:'TYPE BIJKOMENDE TEKST IN, OM TE STOPPEN'
[14]     'TYPE *STOP* BIJ HET BEGIN VAN EEN NIEUWE REGEL'
[15]     →INX
[16]     WIJZ:WIJZIG
[17]     →KIES
[18]     AF:DRUKT
[19]     →KIES
[20]     STOP:'EINDE TEKSTVERWERKING'

```

▽

Het invoeren van een nieuwe tekst kan gestopt worden door het typen van *STOP* bij het begin van een nieuwe regel. Na elke regel wordt automatisch een spatie in de tekst gevoegd om het woord aan het eind gescheiden te houden van het woord aan het begin van de volgende regel tekst. Deze regels blijven geldig bij het invoeren van aanvullende tekst aan het einde van een reeds ingevoerde tekst. Als voorbeeld voeren we een tekst in:

VOER APL TEKST IN

KIES UIT DE VOLGENDE MOGELIJKHEDEN:

1. INBRENGEN NIEUWE TEKST
2. BIJVOEGEN NIEUWE TEKST
3. WIJZIGEN VAN TEKST
4. AFDRUKKEN VAN TEKST
5. STOP

1

TYPE TEKST IN, OM TE STOPPEN TYPE *STOP* BIJ HET
BEGIN VAN EEN NIEUWE REGEL

ALHOEWEL APL LANG BESCHOUWD WERD ALKS EEN TAAL VOOR
WISKUNDIGEN EN TECHNICI, HEEFT HET IN DE LAATSTE TIEN
JAAR VASTE VOET GEKREGEN IN MIDDELGROTE TOT GROOTE
BEDRIJVEN OP GROTE COMPUTERS EN OOK IN KLEINERE BEDRIJVEN M
DE MICROCOMPUTER.

STOP

KIES UIT DE VOLGENDE MOGELIJKHEDEN:

1. INBRENGEN NIEUWE TEKST
2. BIJVOEGEN NIEUWE TEKST
3. WIJZIGEN VAN TEKST
4. AFDRUKKEN VAN TEKST
5. STOP

5

EINDE TEKSTVERWERKING

Omdat de tekst in een globale variabele is opgeslagen, kunnen we nagaan uit hoeveel karakters de tekst bestaat. Om de tekst eventueel onder een andere naam te bewaren kunnen we daar een geschikte variabelenaam voor uitkiezen.

De twee hulpfuncties voeren een iets moeilijker opdracht uit. De eerste functie heeft de naam *WIJZIG*, en de tweede functie heeft de naam *DRUK*. Beide gebruiken als basis voor verwerking de globale variabele met de tekst. De eerste functie kunnen we in enkele te volgen stappen beschrijven. De gebruiker moet de mogelijkheid hebben een woord in de tekst op te zoeken. Dit woord noemen we het referentiewoord. De functie zoekt het gedeelte van de tekst op waarin het referentiewoord voorkomt. Indien het woord niet aanwezig is moet dit worden gemeld. Een gevonden woord mag worden vervangen door een nieuw woord of een reeks woorden. Het referentiewoord wordt in elk geval uit de tekst verwijderd. Indien het echter in de tekst moet blijven staan, dan moet het opnieuw in de nieuwe reeks van woorden worden ingevoerd. In het kort komt het hier op neer dat een foutief geschreven woord het referentiewoord is, en dat de juiste versie het vervangende gedeelte vormt. Is na een bepaald referentiewoord iets vergeten, dan voert men het referentiewoord in als onderdeel van het vervangende deel. Er moet ook rekening gehouden worden met het feit dat het door de gebruiker gekozen referentiewoord verschillende keren in de tekst kan voorkomen. De mogelijkheid moet worden geboden om te zoeken tot het correcte gedeelte van de tekst getoond is om de verandering te kunnen aanbrengen. Deze functie is in zijn geheel:

```

▽ WIJZIG;G;A;TT;POS;I;WOORD;VAN;TOT
[1]  START: 'TYPE IN HET OP TE ZOEKEN WOORD'
[2]  G←pTEKST Δ A←20+□
[3]  POS←0,TT/(1pTEKST)×TT+7>TT←' ,.;?!'1TEKST
[4]  POS←POS[ΔPOS]Δ I←1
[5]  ZOEK:→((pPOS)<I+1)/EIND
[6]  →(POS[I]>pTEKST)/ZOEK
[7]  WOORD←20pTEKST[POS[I-1]+1-1+POS[I]-POS[I-1]],20p'
[8]  →ZOEK ALS 20z+/WOORD=A
[9]  VAN←[ /1,POS[I-1]-20
[10] TOT←(1/(VAN+60),POS[pPOS])-VAN
[11] TEKST[VAN+1TOT]
[12] 'WIJZIGING HIER ?'
[13] →ZOEK ALS 'N'=1+□
[14] 'NIEUW WOORD OF TEKSTGEDEELTE ?'
[15] TEKST←TEKST[1POS[I-1]],□,' ',POS[I]+TEKST
[16] NWIJ: 'ZIJN ER NOG WIJZIGINGEN ?'
[17] →START ALS 'J'=1+□
[18] →0
[19] EIND:→NWIJ Δ ←'KOMT NIET IN TEKST VOOR ...'

```

▽

Enige verklaring zal hier wel noodzakelijk zijn. Op regel [2] bepalen we dat het woord maximaal twintig karakters lang kan zijn. Op de volgende regel worden twee instructies geschreven die te maken hebben met de posities in de vector met tekst waar een woord eindigt. De gesorteerde vector *POS* geeft dan in oplopende volgorde aan waar woorden eindigen. Een woord staat dus altijd tussen twee opeenvolgende berekende posities, uitgezonderd wanneer twee spaties, of een leesteken en een spatie, elkaar opvolgen. In deze laatste gevallen zullen we daarmee bij het opzoeken rekening moeten houden. Op regel drie bepalen we de vector *POS*. Op de volgende twee regels controleren we of er geen referenties zijn naar posities die buiten de tekst vallen. Is dit het geval, dan is het woord niet meer in de tekst terug te vinden. Op regelnummer zeven lichten we een woord uit de tekst tussen twee posities, en op de volgende regel wordt bepaald of het woord wel het gevraagde woord is. Zoniet, dan wordt verder gezocht, anders moet het tekstgedeelte waar het woord in voorkomt aan de gebruiker worden getoond.

Voor dit laatste berekenen we ongeveer 60 posities rondom het woord, er op lettende dat het gedeelte tekst maar vanaf het begin van de gehele tekst kan worden weergegeven. Ook het eind van de te tonen regel mag niet buiten de tekst zelf vallen. Op regel [10] wordt dan het gedeelte tekst getoond. De rest van de functie spreekt voor zichzelf. Het nieuwe gedeelte tekst wordt tussen de oude tekst gevoegd waar het referentiewoord is weggelaten (*POS[I]↓*). Een paar voorbeelden met de ingevoerde tekst zijn:

TYPE IN HET OP TE ZOEKEN WOORD

ALKS

LANG BESCHOUWD WERD ALKS EEN TAAL VOOR WISKUNDIGEN EN TECHNIEK
WIJZIGING HIER ?

JA

NIEUW WOORD OF TEKSTGEDEELTE ?

ALS

ZIJN ER NOG WIJZIGINGEN ?

JA

TYPE IN HET OP TE ZOEKEN WOORD

GROOTE

IN MIDDELGROTE TOT GROOTE BEDRIJVEN OP GROTE COMPUTERS EN O
WIJZIGING HIER ?

JA

NIEUW WOORD OF TEKSTGEDEELTE ?

GROTE

ZIJN ER NOG WIJZIGINGEN ?

NEEN

In de hulpfunctie *DRUK* laten we de tekst volgens een maximum aantal karakters per regel afdrukken. Daarna geven we de keuze de tekst ofwel met ofwel zonder

rechter kantlijn te laten afdrukken. We stellen daarom een keuzemenu samen en vragen naar relevante informatie. De functie in zijn geheel met de verwerkingen van de mogelijkheden is dan:

```

V DRUKT;K;R;POS;LK;TKST;Z;L;G;EXP;PS
[1] 'KIES UIT'
[2] '1 CONTINUE AFDRUK'
[3] '2 FORMAAT AFDRUK'
[4] '3 STOP AFDRUKKEN'
[5] →0 ALS '3'=1↑Q←□
[6] →FORM ALS '2'=1↑Q
[7] TEKST Δ□ ← 6 1 ρ' '
[8] →1
[9] FORM:K←□Δ□ ←'HOEVEEL KARAKTERS PER REGEL?'
[10] R←'J'=1↑□Δ□ ←'MOET DE TEKST RECHTS AANSLUITEN?'
[11] POS←1 Δ G←ρTEKST
[12] LIJN:→0 ALS POS>G
[13] →DRUKAF ALS G<LK←POS+K-1
[14] I←1
[15] VERKORT:→VERKORT ALS TEKST[LK←POS+K-I+I+1]≠' '
[16] →MARGE ALS R=1
[17] DRUKAF:LK←⌊G,LK
[18] ' ',TEKST[POS],TEKST[POS+1L←LK-POS]
[19] →LIJN Δ POS←LK+1
[20] MARGE:TKST←TEKST[POS],TEKST[POS+1L←LK-POS+1]
[21] EXP←(TKST=' ')+EXP←(L+1)ρ0
[22] EXP←(L+1)ρI+0 Δ PS←EXP/EXP×1ρEXP
[23] VOEGIN:→DRUKF ALS (K-ρTKST)<I+I+1
[24] EXP←EXP[1PS[Z]],1,PS[Z←?ρPS]←EXP
[25] PS←PS+PS≥PS[Z]
[26] →VOEGIN
[27] DRUKF:TKST←(∼EXP)\TKST
[28] →LIJN Δ POS←LK+1 Δ□ ←' ',TKST
V

```

De tekst wordt dus verkort tot de positie waar een woord of een zin eindigt. Zodra het verkorte tekstgedeelte in de vector *TKST* bepaald is, kan de tekst worden verspreid over de breedte van de regel zoals bepaald door het aantal karakters per regel. We doen het opvullen met spaties willekeurig zodat niet op elke regel telkens in het begin, midden of eind, een aantal spaties de tekstlijn moeten opvullen. Hiervoor maken we gebruik van het teken *?*, en dan een logische vector *EXP* om via de uitbreidingsfunctie de gewenste afdruk te geven.

Als voorbeeld vragen we de ingevoerde tekst op een breedte van 45 karakters af te drukken met een rechte kantlijn rechts:

KIES UIT

1 CONTINUE AFDRUK

2 FORMAAT AFDRUK

3 STOP AFDRUKKEN

2

HOEVEEL KARAKTERS PER REGEL?

45

MOET DE TEKST RECHTS AANSLUITEN?

JA

ALHOEWEL APL LANG BESCHOUWD WERD ALS EEN
TAAL VOOR WISKUNDIGEN EN TECHNICI, HEEFT HET
IN DE LAATSTE TIEN JAAR VASTE VOET GEKREGEN
IN MIDDELGROTE TOT GROTE BEDRIJVEN OP GROTE
COMPUTERS EN OOK IN KLEINERE BEDRIJVEN MET DE
MICROCOMPUTER.

8.4.2 Huishoudelijke budgettering

Om verschillende gedefinieerde functies een grotere toepasbaarheid te geven, is het soms aan te raden een extra functie te schrijven, die door middel van een paar globale variabelen het specifieke probleem of de specifieke toestand beschrijft. In het voorbeeld in hoofdstuk 5 over het opmaken van een budget en het bijhouden van uitgaven hebben we enkele posten gekozen die hiervoor in aanmerking komen. Om het aantal posten en zelfs het aantal maanden te wijzigen, zou men veranderingen in een gedefinieerde functie moeten aanbrengen. Door het schrijven van een extra functie die maar één keer hoeft te worden gebruikt, maakt men het mogelijk individueel de dimensies van het probleem te bepalen. We definiëren dus een functie die de naam *INIT* heeft. Daarin bepalen we de dimensies van alle vectoren, de matrix en de alfanumerieke gegevens zoals die in de uitvoer moeten voorkomen. Deze functie heeft dus als doel het initialiseren van de gegevens:

```

▽ INIT;P
[1] NM←⊂⌈Δ⌋ ←'HOEVEEL MAANDEN WORDEN GEBUDGETTEERD?'
[2] BUDGM←(12,(NM+3))ρ0
[3] P←(13↑'VOEDING'),(13↑'KLEDING'),(13↑'ENERGIE')
[4] P←P,(13↑'HYPOTHEEK'),(13↑'LENINGEN'),(13↑'VERZEKERING')
[5] P←P,(13↑'AUTO/REIZEN'),(13↑'TELEFOON'),(13↑'HOBBY')
[6] P←P,(13↑'ONVOORZIEN'),(13↑'INKOMSTEN'),(13↑'BALANS')
[7] POSTEN← 12 13 ρP
[8] MAANDEN← 12 3 ρ'JANFEBMARAPRMEIJUNJUKAUGSEPOCTNOVDEC'
[9] NO←⊂⌈Δ⌋ ←'WAT IS HET NUMMER VAN MAAND 1 (1 - 12)?'
[10] NO←NO-12×(NO←(NO-1)+1NM)>12

```

▽

Deze functie heeft geen uitvoer van gegevens nodig omdat deze onder de globale variabelnamen beschikbaar zijn. De functie wordt als volgt gebruikt:

```

      INIT
HOEVEEL MAANDEN WORDEN GEBUDGETTEERD?
2
WAT IS HET NUMMER VAN MAAND 1 (1 - 12)?
11

      MAANDEN[NO;]
NOV
DEC
      ρPOSTEN
12 13
      ρBUDGM
12 5

```

De functie met de analyse van het budget gebruikt deze gegevens en moet aan enkele eisen voldoen. Ten eerste moet de gebruiker een reeks mogelijkheden krijgen om een reeks van verschillende gegevens in te voeren en te evalueren. Daartoe zal gebruik worden gemaakt van een hoofdmenu. Ten tweede moeten bepaalde uitkomsten niet alleen in tabelvorm kunnen worden weergegeven, maar liefst ook grafisch.

De gebruiker moet altijd goed weten waar hij aan toe is bij het uitvoeren van een functie. Niet alleen zal het belangrijk zijn te weten wat de mogelijkheden tot invoer zijn, of de functie niladisch, monadisch of dyadisch is, maar ook hoe de uitvoer van de functie er uit zal zien. Uit de functie *INIT* krijgen we de matrix *BUDGM*. Het aantal rijen in deze matrix komt overeen met de posten van uitgaven, een post voor inkomsten en één voor de balans. Het aantal kolommen werd bepaald door het aantal te budgetteren maanden, plus drie. Deze drie kolommen zijn respectievelijk gereserveerd voor het vaste maandbudget, het verschil tussen de posten van een bepaalde maand, en die van het vaste budget en het procentuele verschil. De gebruiker moet ook weten dat dit de laatste drie kolommen in de matrix zijn.

Om diverse berekeningen tussen de kolommen toe te laten zal de gebruiker de keuze hebben uit enkele mogelijkheden. Er moet bijvoorbeeld rekening worden gehouden met de mogelijkheid dat een gebruiker negen kolommen (voor zes te budgetteren maanden) in de matrix heeft, of maar vier kolommen (voor één maand). De gedefinieerde functie *BUDGET* heeft, zoals hieronder blijkt, een hoge vorm van interactie om de gebruiker de juiste berekeningen te laten verrichten, de goede vorm van een rapport te kiezen, en het juiste histogram te gebruiken:

```

▽ BUDGET;HS;Q;I;K;UIT
[1] HS← 10 3 Δ□ ←'KIES UIT:'
[2] ' 1. INVÖEREN VAST MAAND BUDGET'
[3] ' 2. INVOEREN UITGAVEN/INKOMSTEN 1 MAAND'
[4] ' 3. INVOEGING NIEUWE MAAND'
[5] ' 4. BEREKENEN/AFDRUKKEN BUDGT'
[6] ' 5. TEKENEN HISTOGRAM'
[7] Q←2Δ□ ←' 6. EINDE VAN BUDGET ANALYSE'
[8] →((Q=1),(Q=2),(Q=3))/VST,MND,NEW
[9] →((Q=4),(Q=5),(Q=6))/BER,HIS,END
[10] VST:'GEEF 1 WAARDE VOOR ELKE POST IN VECTOR VORM'
[11] →1 Δ BUDGM[;NM+1]←12ρ2Δ□
[12] MND:K←2Δ□ ←'WELKE MAAND?' Δ I←0
[13] PERΔPOST:→BAL ALS 12=I←I+1
[14] BUDGM[I;K]←2Δ□ ←'BEDRAG VOOR ',POSTEN[I;]
[15] →PERΔPOST
[16] BAL:→1 Δ BUDGM[12;K]←BUDGM[11;K]-+/BUDGM[;10;K]
[17] NEW:'MAAND ',MAANDEN[NO[1];],' VERVALT...'
[18] NO←NO-12×(NO←NO+1)>12
[19] 'MAAND ',MAANDEN[NO[NM];],' IS NIEUWE MAAND'
[20] →1 Δ BUDGM[;NM]←0 Δ BUDGM[;1NM]←BUDGM[;1+1NM]
[21] BER:K←2Δ□ ←'NUMMER VAN MAAND TER VERGELIJKING BUDGET?'
[22] BUDGM[;NM+2]←BUDGM[;K]-BUDGM[;NM+1]
[23] BUDGM[;NM+3]←100×BUDGM[;NM+2]÷BUDGM[;NM+1]
[24] ' ' Δ□ ←' ANALYSE BUDGET ' Δ□ ← 2 1 ρ'
[25] UIT←(4ρ' '), 'POSTEN ',MAANDEN[NO;],[2](NM,5)ρ'
[26] I←0 Δ□ ←' ' Δ□ ←UIT,' BUDGET VERSCHIL °/°'
[27] PΔP:→1 ALS 13=I←I+1
[28] →PΔP Δ□ ←POSTEN[I;], 8 0 ▽BUDGM[I;]
[29] HIS:'KIES UIT:'
[30] ' 1. HISTOGRAM PER POST VOOR 1 MAAND'
[31] ' 2. HISTOGRAM PER MAAND VOOR 1 POST'
[32] →PP ALS 2=2Δ□
[33] HS HISTOGRAM BUDGM[;K←2Δ□]Δ□ ←'WELKE MAAND?'
[34] →1
[35] PP:HS HISTOGRAM BUDGM[K←2Δ□;1NM]Δ□ ←'WELKE POST'
[36] →1
[37] END:'EINDE BUDGET ANALYSE' Δ□ ← 2 1 ρ'

```

▽

Het gebruik van een hoofdmenu maakt het overzicht van de mogelijkheden iets duidelijker. Na elke selectie zijn de verdere opties duidelijk toegelicht, zodat er geen verkeerd te interpreteren instructies aan de computer worden gegeven. We bespreken het gebruik van de functie aan de hand van enkele voorbeelden. We veronderstellen dat de budgetmatrix uit de vaste twaalf rijen en vijf kolommen

bestaat zodat er twee te budgetteren maanden zijn. De gegevens in kolom één zijn bekend voor de maand en voor het vaste budget. We voeren de gegevens voor de tweede maand in en we vragen vervolgens naar de berekening van het verschil tussen de bedragen van de tweede maand en het vaste budget:

*BUDGET**KIES UIT:*

1. *INVOEREN VAST MAAND BUDGET*
2. *INVOEREN UITGAVEN/INKOMSTEN 1 MAAND*
3. *INVOEGING NIEUWE MAAND*
4. *BEREKENEN/AFDRUKKEN BUDGET*
5. *TEKENEN HISTOGRAM*
6. *EINDE VAN BUDGET ANALYSE*

2

WELKE MAAND?

2

BEDRAG VOOR VOEDING

920

BEDRAG VOOR KLEDING

260

BEDRAG VOOR ENERGIE

225

BEDRAG VOOR HYPOTHEEK

1200

BEDRAG VOOR LENINGEN

260

BEDRAG VOOR VERZEKERINGEN

210

BEDRAG VOOR AUTO/REIZEN

120

BEDRAG VOOR TELEFOON

0

BEDRAG VOOR HOBBY

110

BEDRAG VOOR ONVOORZIEN

250

BEDRAG VOOR INKOMSTEN

3922

KIES UIT:

1. *INVOEREN VAST MAAND BUDGET*
2. *INVOEREN UITGAVEN/INKOMSTEN 1 MAAND*
3. *INVOEGING NIEUWE MAAND*
4. *BEREKENEN/AFDRUKKEN BUDGET*
5. *TEKENEN HISTOGRAM*
6. *EINDE VAN BUDGET ANALYSE*

4

De resultaten zijn in een tabel als volgt weergegeven:

KIES UIT:

1. INVOEREN VAST MAAND BUDGET
2. INVOEREN UITGAVEN/INKOMSTEN 1 MAAND
3. INVOEGING NIEUWE MAAND
4. BEREKENEN/AFDRUKKEN BUDGET
5. TEKENEN HISTOGRAM
6. EINDE VAN BUDGET ANALYSE

4

NUMMER VAN MAAND TER VERGELIJKING BUDGET?

2

ANALYSE BUDGET

POSTEN	NOV	DEC	BUDGET	VERSCHIL	o/o
VOEDING	900	920	900	20	2
KLEDING	200	260	200	60	30
ENERGIE	225	225	225	0	0
HYPOTHEEK	1200	1200	1200	0	0
LENINGEN	260	260	260	0	0
VERZEKERINGEN	100	210	220	-10	-5
AUTO/REIZEN	180	120	160	-40	-25
TELEFOON	200	0	80	-80	-100
HOBBY	100	110	100	10	10
ONVOORZIEN	300	250	250	0	0
INKOMSTEN	2900	3922	3900	22	1
REST	765	367	305	62	20

Als tweede voorbeeld beginnen we wat de uitvoer betreft meteen met de keuze voor een histogram. Uit het submenu kiezen we per post van maand twee (DEC) een histogram:

BUDGET

KIES UIT:

1. INVOEREN VAST MAAND BUDGET
2. INVOEREN UITGAVEN/INKOMSTEN 1 MAAND
3. INVOEGING NIEUWE MAAND
4. BEREKENEN/AFDRUKKEN BUDGET
5. TEKENEN HISTOGRAM
6. EINDE VAN BUDGET ANALYSE

5

KIES UIT:

1. HISTOGRAM PER POST VOOR 1 MAAND
2. HISTOGRAM PER MAAND VOOR 1 POST

1

WELKE MAAND?

2

											*	3922
											*	3530
											*	3138
											*	2745
											*	2353
											*	1961
											*	1569
			*								*	1177
*			*								*	784.4
*	*	*	*	*	*	*	*	*	*	*	*	392.2
1	2	3	4	5	6	7	8	9	10	11	12	

Als derde voorbeeld voeren we een nieuwe maand in. Dit wil zeggen, niet de data voor die maand, maar een referentie naar een maand als nieuwe kolom in de matrix. Dit betekent dat te budgetteren maanden naar links worden 'verschoven'. We volgen de functie vanaf de regelnaam *NEW*. Maand twaalf (DEC) neemt de plaats in van maand elf (NOV). Maand één (JAN) verschijnt in de tweede kolom. Het effect hiervan is dat de maand november uit de tabel geschoven werd. Het geheel inclusief het rapport ziet er als volgt uit:

KIES UIT:

1. INVOEREN VAST MAAND BUDGET
2. INVOEREN UITGAVEN/INKOMSTEN 1 MAAND
3. INVOEGING NIEUWE MAAND
4. BEREKENEN/AFDRUKKEN BUDGET
5. TEKENEN HISTOGRAM
6. EINDE VAN BUDGET ANALYSE

3
MAAND NOV VERVALT...

MAAND JAN IS NIEUWE MAAND

KIES UIT:

1. INVOEREN VAST MAAND BUDGET
2. INVOEREN UITGAVEN/INKOMSTEN 1 MAAND
3. INVOEGING NIEUWE MAAND
4. BEREKENEN/AFDRUKKEN BUDGET
5. TEKENEN HISTOGRAM
6. EINDE VAN BUDGET ANALYSE

4

NUMMER VAN MAAND TER VERGELIJKING BUDGET?

2

ANALYSE BUDGET						
POSTEN	DEC	JAN	BUDGET	VERSCHIL	o/o	
VOEDING	920	0	900	-900	-100	
KLEDING	260	0	200	-200	-100	

ENERGIE	225	0	225	~ 225	~ 100
HYPOTHEEK	1200	0	1200	~ 1200	~ 100
LENINGEN	260	0	260	~ 260	~ 100
VERZEKERINGEN	210	0	220	~ 220	~ 100
AUTO/REIZEN	120	0	160	~ 160	~ 100
TELEFOON	0	0	80	~ 80	~ 100
HOBBY	110	0	100	~ 100	~ 100
ONVOORZIEN	250	0	250	~ 250	~ 100
INKOMSTEN	3922	0	3900	~ 3900	~ 100
REST	367	0	305	~ 305	~ 100

Alhoewel de functie nogal gebruikersvriendelijk lijkt te zijn, is er helemaal geen foutencontrole geprogrammeerd. De lezer kan met betrekking tot de oefeningen controles tussenvoegen zonder dat er verder grote wijzigingen in de functie worden aangebracht. De laatste opmerking betreft de vector *HS* die bij de histogram-functie wordt gebruikt. De gebruiker kan de getallen in deze vector naar willekeur wijzigen, of kan interactief aan de gebruiker laten vragen wat de grootte van de grafiek moet zijn. Dit in verband met de grootte van bijvoorbeeld het beeldscherm.

8.4.3 Een statistische functie

Bij de vorige toepassing hadden we wellicht gebruik kunnen maken van enkele statistische verwerkingen om de analyse van uitgaven en budget nog verder te vergemakkelijken. De meeste APL-systemen hebben echter een 'bibliotheek' waar statistische functies in zijn opgenomen en die door elke gebruiker kunnen worden gecopieerd. Als voorbeeld van een statistische functie nemen we hier een functie die in eerste instantie test of het rechtse gegeven van de functie een vector is en, indien dit het geval is, of het wel een numerieke vector is. Daarna worden het minimum, het maximum, het gemiddelde, de variatie en de standaarddeviatie berekend. Deze functie, gevolgd door een toepassing, ziet er als volgt uit:

```

▽ STAT X
[1] 'LENGTE INVOER (≤1)' FOUT 1=ρ,X←ρX
[2] 'AANTAL GETALLEN'           ',ρρX
[3] 'HOOGSTE GETAL'             ',ρ[ /X
[4] 'LAAGSTE GETAL'             ',ρ[ /X
[5] 'GEMIDDELDE'                ',ρGEM←(+ /X)÷ρX
[6] 'VARIATIE'                  ',ρVAR←+ / ((X-GEM)*2)÷(ρX)-
[7] 'STANDAARD DEVIATIE'        ',ρSD←VAR*0.5

▽
STAT BUDGM[;1]
AANTAL GETALLEN      12
HOOGSTE GETAL        3922

```


LAAGSTE GETAL	0
GEMIDDELDE	653.6666667
VARIATIE	1181339.697
STANDAARD DEVIATIE	1086.89452

Als toepassing hebben we de uitgaven van de eerste maand genomen uit het voorbeeld van de functie *BUDGET*.

8.4.4 Voorbeeld van computer ondersteund onderwijs (COO)

Als men op een bepaalde scholengemeenschap voor ondersteuning van het onderwijs de computer wil gaan gebruiken, dan denkt men meestal aan computers ten behoeve van de administratie en in mindere mate aan het lesgeven met de computer. Ook op de Nederlandse markt zijn er reeds enkele goede onderwijs-systemen die tot doel hebben het individueel onderwijs nog meer te bevorderen. Hierbij wordt de computer ingeschakeld om bepaalde schriftelijke oefeningen op een geautomatiseerde manier te verzekeren.

Als voorbeeld nemen we een standaard applicatie van COO. Aan de student wordt een tekst gegeven die aandachtig moet worden gelezen. Als de tekst bestudeerd is worden er over de inhoud enkele vragen gesteld. De student beantwoordt deze vragen en krijgt te zien of het antwoord goed of fout is.

Om dit via een *APL*-functie te laten gebeuren, bouwen we het geheel modulair op. We maken een tekst aan op een willekeurige wijze, met behulp van de tekstverwerkende functie of als variabele.

We noemen de hoofdfunctie *LEER*, waarvan we hier de eerste vijf regels weer-geven:

```

▽ LEER V;DL;A;I;G;NV
[1] 'LEES DE TEKST HIERONDER AANDACHTIG EN'
[2] ' ' Δ 'ANTWOORD DAARNA DE VRAGEN'
[3] V
[4] DL←ΔDL 5
[5] KIESV:I←G←0 Δ NV←3?(1+pVR)

```

Op regel [4] wordt de tekst aan de student getoond. Op de volgende regel laten we de functie enige tijd wachten door middel van een systeemvariabele *delay* die we in het volgende hoofdstuk nader toelichten. We definiëren verder een functie *VRAGEN* waarin we een matrix *VR* specificeren met een vijftal vragen over de tekst. De reden om de matrix *VR* in een functie aan te maken is dat het later gemakkelijker is om eventuele veranderingen aan te brengen. De functie met vragen is bijvoorbeeld:

```

▽ VRAGEN
[1] I←0 Δ VR← 0 50 ρ' '
[2] 'STEL VRAAG ',▽I←I+1
[3] VR←VR,[1]50↑▽
[4] →2 ALS I<5
▽

```

We maken op dezelfde manier een matrix met antwoorden, dus ook via een functie die we *ANTWOORD* noemen. Hierin nemen we echter een driedimensionele matrix waar de eerste dimensie het aantal vragen aangeeft, de tweede dimensie twee mogelijke goede antwoorden per vraag, en de derde dimensie de lengte van het antwoord. De matrix zelf noemen we *AA* en de functie is:

```

▽ ANTWOORD;I;J
[1] I←0
[2] J←0 Δ I←I+1
[3] 'GEEF ANTWOORD ',(▽J+J+1),' VOOR VRAAG ',▽I
[4] AA[I;J;]←20↑▽
[5] →3 ALS J<2
[6] →2 ALS I<5
▽

```

Het vervolg van de functie *LEER* verloopt dan verder volgens een scenario waar een vraag wordt gesteld, een antwoord wordt gegeven, het antwoord wordt geëvalueerd en de score wordt bijgehouden. Als variant laten we het systeem drie willekeurige vragen stellen uit de reeks van vijf. De rest van de functie *LEER* is als volgt:

```

[6] VRAAG:→GEDAAN ALS 3<I+I+1
[7] A←20↑▽Δ▽ +VR[NV[I];]Δ▽ + 'VRAAG ',▽I Δ ' '
[8] →GOED ALS 20=+/A=AA[NV[I];1;]
[9] →GOED ALS 20=+/A=AA[NV[I];2;]
[10] 'FOUT: HET GOEDE ANTWOORD IS ',AA[NV[I];1;]
[11] →VRAAG
[12] GOED: 'GOED ZO ' Δ G←G+1
[13] →VRAAG
[14] GEDAAN:(▽G),' ANTWOORDEN WAREN CORRECT'
[15] 'WILT U NOG ENKELE VRAGEN BEANTWOORDEN?'
[16] →KIESV ALS 'J'=1↑▽
▽

```

Het uitvoeren van de bovenstaande functie is voorafgegaan door de uitvoering van de twee functies die de vragen en antwoorden bepalen. Het geheel kan enkele malen naar willekeur worden uitgevoerd, waarvan hier een eenvoudig voorbeeld:

LEER TEKST
LEES DE TEKST HIERONDER AANDACHTIG EN
ANTWOORD DAARNA DE VRAGEN

DE COMPUTERTAAL APL KAN WORDEN GEBRUIKT DOOR MENSEN MET
VERSCHILLENDE ACHTERGRONDEN EN NIVEAUS VAN OPLEI
DING. APL IS EEN VERTALER DIE DE INSTRUKTIES REG
EL NA REGEL VERTAALT EN UITVOEKT. DIT IS IN TEGE
NSTELLING MET EEN 'COMPILER'TAAL WAAR EERST ALLE
INSTRUKTIES DIENEN TE WORDEN OMGEZET NAAR MACHINE
TAAL EN DAN PAS WORDEN UITGEVOERD.

VRAAG 1

APL IS EEN OPLEIDING (JA OF NEEN)

NEEN

GOED ZO

VRAAG 2

BEGRIJPT DE COMPUTER APL OF MACHINETAAL ?

APL

FOUT: HET GOEDE ANTWOORD IS MACHINETAAL

VRAAG 3

IS APL EEN VERTALER OF EEN COMPILER TAAL

VERTALER

GOED ZO

2 ANTWOORDEN WAREN CORRECT

WILT U NOG ENKELE VRAGEN BEANTWOORDEN?

NEEN

8.4.5 Voorbeeld van een financiële analyse

Als laatste voorbeeld hebben we voor een toepassing gekozen die enig verband houdt met het budgetteringsvoorbeeld, maar toch een grotere flexibiliteit aan de gebruiker geeft. We stellen het op te lossen probleem als volgt voor. Bij een financiële analyse moet men de mogelijkheid hebben een standaardfunctie als een hulpmiddel te gebruiken om met bepaalde instructies een financiële rapportering te kunnen maken. Deze instructies zelf moeten eenvoudig en gemakkelijk aan te passen zijn aan individuele omstandigheden. Dit betekent dat men de vorm van het rapport en ook de uit te voeren bewerking, op een bepaalde manier zal kunnen opgeven zonder iets van programmeren af te weten. Omdat dit een verkort voorbeeld is van hetgeen bij grotere computers en software pakketten gebruikt wordt, laten we onze APL-functie enkele meest noodzakelijke instructies herkennen.

We definiëren rapporten altijd in de vorm van een matrix waarbij elke rij en elke kolom een door de gebruiker te kiezen naam krijgt. We geven deze namen met sleutelwoorden of met de een code als volgt aan:

RIJ:VERKOOP
RIJ:ONKOSTEN
RIJ:BRΔWINST
KOL:JAAR1
KOL:JAAR2

Hierbij spreken we af dat een sleutelwoord bestaat uit drie letters, onmiddellijk gevolgd door een dubbele punt. Bij de definitie van namen gebruiken we maximaal acht posities. Verder kunnen we bepalen wat voor berekeningen er moeten worden uitgevoerd om tot een goed rapport te komen. Bij financiële planning zijn voor een rij of kolom meestal enkele gegevens bekend, en moeten de cijfers voor andere rijen en/of kolommen aan de hand van de bekende cijfers worden berekend. We hebben dus twee mogelijkheden: ofwel worden cijfers gegeven aan een bepaalde rij (of kolom) via de naam van die rij (of kolom), ofwel deze worden via eenvoudige berekeningen bepaald. Enkele voorbeelden maken dit duidelijk:

BER:VERKOOP+1200
BER:ONKOSTEN+600 700
BER:BRΔWINST+VERKOOP-ONKOSTEN

We specificeren de rij met verkoopcijfers als 1200 gulden. Dit getal komt dan onder de kolom van het eerste jaar *en* onder de kolom van het tweede jaar te staan. De kosten van 600 en 700 gulden zijn respectievelijk voor het eerste en tweede jaar. De bruto winst wordt voor de totale rij berekend.

Om het rapport verzorgd afgedrukt te krijgen kan men gebruik maken van de volgende twee sleutelwoorden:

HFD:FINANCIEEL RAPPORT *ABC BV.*
HFD: (JAAR1=1982 / JAAR2=1983)
DAT:

waarbij *HFD* betekent dat de daaropvolgende tekst als titel wordt afgedrukt, en waarbij *DAT* betekent dat de datum wordt afgedrukt. Deze laatste instructie heeft na de dubbele punt geen gegevens nodig.

Aangezien bij grotere rapporten soms niet alle rijen of kolommen hoeven te worden weergegeven, bestaat de mogelijkheid via de instructie met sleutelwoord *REP* die rijen en kolommen op te geven die in het rapport moeten komen. De namen van deze rijen en kolommen moeten zijn gescheiden door een spatie. Zodra het einde van de instructies is bereikt worden alleen de relevante gegevens afgedrukt.

We schrijven alle instructies die moeten dienen voor het construeren van een rapport in een functie die we *MODEL* noemen:


```

      ∇  MODEL
[1]  KOL:1981
[2]  KOL:1982
[3]  KOL:1983
[4]  RIJ:INKOMEN
[5]  RIJ:UITGAVEN
[6]  RIJ:BRΔWINST
[7]  RIJ:KOSTEN
[8]  RIJ:VERSCHIL
[9]  RIJ:BELAST
[10] RIJ:NTΔWINST
[11] HFD:FINANCIELE PLANNING
[12] DAT:
[13] BER:INKOMEN←1000,1120,1409
[14] BER:UITGAVEN← 400 600 800
[15] BER:KOSTEN←3p100
[16] BER:BRΔWINST←INKOMEN-UITGAVEN
[17] BER:VERSCHIL←BRΔWINST-KOSTEN
[18] BER:BELAST←0.49×VERSCHIL
[19] BER:NTΔWINST←VERSCHIL-BELAST
[20] REP:INKOMEN UITGAVEN BRΔWINST KOSTEN
[21] REP:VERSCHIL BELAST NTΔWINST
[22] REP: 1981 1982 1983
[23] END:
      ∇

```

Deze functie kunnen we omzetten naar een variabele zodat de hoofdfunctie deze gegevens zal kunnen verwerken. Dit omzetten naar een variabele kunnen we doen door als volgt gebruik te maken van de systeemfunctie $\square CR$:

$F \leftarrow \square CR \text{ 'MODEL'}$

Het resultaat van deze functie is een matrix met alle regels van de functie tussen aanhalingstekens, inclusief de naam van de functie. De regelnummers zelf en het teken ∇ zijn weggelaten. De matrix is een alfanumerieke vector waarvan het aantal rijen gelijk is aan het aantal regels in de functie plus één, en waarvan het aantal kolommen gelijk is aan het aantal karakters op de langste regel.

Deze matrix bevat nu alle informatie over het te schrijven rapport. Het verwerken van deze informatie doen we via een gedefinieerde functie *FINAN* door eerst na te gaan of alles logisch en toegestaan is, en dan over te gaan tot het samenstellen van de resultaten en het rapport:

```

V  FINAN F;I;J;D;CNTR;RIJS;KOLS;Z;BEREKEN;A;P
[1] →GEENΔG ALS 0=+/ρF Δ□ +50ρ'-' Δ□ + 2 1 ρ' '
[2] D←1+ρF←F,[2]' ' Δ I←1 Δ CNTR← 7 3 ρ'KOLRIJHDDTBERREPE
[3] KOLNAAM←RIJNAAM←(08)ρ' ' Δ M← 0 0 ρ0 Δ RIJS←KOLS←10
[4] VOLGENDE:→END ALS(1+ρF)<I+I+1
[5] →FOUT1 ALS F[I;4]≠': '
[6] →GOED ALS(ρCNTR)[1]≥J←(+ /+ /CNTR°. =F[I;13])13
[7] FOUT1: ' CONTROLE FOUT OF '': ' NIET GEGEVEN IN ' ;F[I;]
[8] →END
[9] GOED:→((J=1),(J=2),(J=3))/KOL,RIJ,HFD
[10] →((J=4),(J=5),(J=6),(J=7))/DAT,BER,REP,END
[11] KOL:→VOLGENDE ALS 0=Z←1 ZOEKΔOP F[I;4+18]
[12] RIJ:→VOLGENDE ALS 0=Z←2 ZOEKΔOP F[I;4+18]
[13] HFD:→VOLGENDE Δ□ +4+F[I;]Δ□ ←' '
[14] DAT: ' DATUM: ',(▼□TS[3]),'/',(▼□TS[2]),'/',(▼□TS[1]
[15] →VOLGENDE Δ□ ←' ' Δ□ +50ρ'-'
[16] BER:→BER1 ALS 0≠+/ρM
[17] M←((1+ρRIJNAAM),1+ρKOLNAAM)ρ0
[18] BER1:A←BEREKEN←10 Δ P←4
[19] ZOEK:→VOERUIT ALS(Ē←P+1)>D
[20] →BIJA ALS 28>'ABCDEFGHIJKLMNPOQRSTUVWXYZΔ',F[I;P]
[21] →BIJBEREK ALS 0=ρA
[22] P←P-1
[23] →MRIJ ALS 1≠Z←4 ZOEKΔOP A
[24] →FOUT2 ALS 1=Z←3 ZOEKΔOP A
[25] BEREKEN←BEREKEN,'M[;',(▼|Z),']'
[26] →ZOEK ALS 0=ρA←10
[27] BIJA:→ZOEK Δ A←A,F[I;P]
[28] MRIJ:BEREKEN←BEREKEN,'M[;',(▼Z),';:]'
[29] →ZOEK ALS 0=ρA←10
[30] BIJBEREK:→ZOEK Δ BEREKEN←BEREKEN,F[I;P]
[31] VOERUIT:→VOLGENDE ΔBBEREKEN
[32] REP:A←10 Δ P←4
[33] VERVOLG:→VOLGENDE ALS D<P←P+1
[34] →VERVOLG ALS(0=ρA)∧F[I;P]=' '
[35] A←A,F[I;P]
[36] →VERVOLG ALS ' '≠A[ρA]
[37] →BRIJ ALS 1≠Z←4 ZOEKΔOP 1←A
[38] →FOUT2 ALS 1=Z←3 ZOEKΔOP 1←A
[39] →VERVOLG Δ A←10 Δ KOLS←KOLS,Z
[40] BRIJ:→VERVOLG Δ A←10 Δ RIJS←RIJS,Z
[41] FOUT2:→EINDE Δ□ ←'DE NAAM <','A,'> IS ONBEKEND '
[42] END:I←0
[43] ' ' Δ□ +(13ρ' ' ),,KOLNAAM[KOLS;],[2]((ρKOLS),2)ρ' '
[44] DRUK:→EINDE ALS(ρRIJS)<I+I+1
[45] ' ' Δ□ +RIJNAAM[RIJS[I];],[10 2 ▼M[RIJS[I];KOLS]]
[46] →DRUK
[47] GEENΔG: 'DE GEGEVENS ZIJN NIET BESCHIKBAAR'
[48] EINDE: 'EINDE ANALYSE' Δ□ ←' '

```


De functie *FINAN* heeft als eerste taak het initialiseren van de verwerkingsmatrices. Alle rijen van de gegevensmatrix worden doorlopen en getoetst op de sleutelwoorden zoals die voorkomen in de vector *CNTR*. Bij het ontbreken van een dubbele punt, of het niet herkennen van het sleutelwoord, wordt meteen een foutmelding gemaakt. Aangezien de gebruiker vrij is rij- en kolomnamen van maximaal acht karakters te bepalen, wordt er een lijst aangelegd van deze namen. Telkens als een naam wordt gebruikt, zoals bij de *KOL*: instructie of bij de *REP*: instructie, doen we beroep op een functie die de naam in de lijst opzoekt. In deze functie laten we een foutmelding toe als de naam moet worden opgezocht en niet wordt gevonden, en we gaan gewoon verder als de naam aan de lijst mag worden toegevoegd. In alle gevallen dient een onderscheid te worden gemaakt tussen het toevoegen aan één van beide lijsten, en het opzoeken in één van beide lijsten. De bewerkingen zijn zeer eenvoudig tot het moment er berekeningen moeten worden uitgevoerd. De gebruiker is namelijk verplicht *APL*-notatie te gebruiken. Enkele toegestane berekeningen zijn:

```
BER:NAAM←1○NAAM←NAAM1
BER:NAAM←(NAAM×2)÷1.5
BER:NAAM1←NAAM2+NAAM3
```

Instructies die niet overeenstemmen met de syntax van *APL* of die namen van rijen of kolommen bevatten die niet in de lijst voorkomen, resulteren in een foutmelding en betekenen de beëindiging van de functie. Dit is duidelijk in:

```
BER:NAAM←=4 16 18
BER:NAAM←4+NAAMZEVEN
BER:NAAM←(NAAM×.4)÷Q←1.11
```

waarbij in de laatste berekening de variabelenaam *Q* niet in de lijst van namen voorkomt. De berekeningen worden karakter na karakter gecontroleerd om te zien of er namen dan wel primitieve functies worden gebruikt. Aangezien alleen maar alfabetische karakters als naam kunnen worden gebruikt is deze test heel eenvoudig. Telkens als een naam is gevonden wordt deze vervangen door een referentie naar de matrix *M*[.]. Alle andere voorkomende tekens worden gewoon overgenomen in een vector met de naam *BEREKEN*. Wanneer de berekening volledig is gecontroleerd wordt deze laatste met de functie Φ uitgevoerd.

Bij de bepaling van welke rijen en kolommen in het rapport moeten verschijnen, maken we van hetzelfde principe van positiebepaling gebruik als bij de functie voor tekstverwerking.

De functie om de namen op te zoeken is:

```

      ▽ Z←ISW ZOEKΔOP VARIABELE;VAR
[1]  Z←0 Δ VAR←8+VARIABELE
[2]  →((ISW=2),(ISW=3),(ISW=4))/TRIJ,NKOL,NRIJ
[3]  →0 Δ KOLNAAM←KOLNAAM,[1]VAR
[4]  TRIJ:→0 Δ RIJNAAM←RIJNAAM,[1]VAR
[5]  NKOL:→FV ALS(1+pKOLNAAM)<Z+Z+1
[6]  →0 ALS 8=+/VAR=KOLNAAM[Z;]
[7]  →NKOL
[8]  NRIJ:→FV ALS(1+pRIJNAAM)<Z+Z+1
[9]  →0 ALS 8=+/VAR=RIJNAAM[Z;]
[10] →NRIJ
[11] FV:Z←1
      ▽

```

Alles is nu klaar voor de uitvoering van de functie *FINAN* en het volgende rapport wordt afgedrukt:

FINAN F←[CR 'MODEL'			

FINANCIELE PLANNING			
DATUM: 1/3/1983			

	1981	1982	1983
INKOMEN	1000.00	1120.00	1409.00
UITGAVEN	400.00	600.00	800.00
BRAWINST	600.00	520.00	609.00
KOSTEN	100.00	100.00	100.00
VERSCHIL	500.00	420.00	509.00
BELAST	245.00	205.80	249.41
NTΔWINST	255.00	214.20	259.59
EINDE ANALYSE			

De lezer kan nu zonder echt te gaan programmeren de gewenste rapporten samenstellen. De functie *FINAN* kan worden gebruikt ter vervanging van de

functie *BUDGET*. Omdat de variabele *M* een globale variabele is kan eventueel met de daarin berekende gegevens de histogramfunctie worden gebruikt.

8.5 Oefeningen

1. Bestudeer de histogramfunctie en breng de volgende veranderingen aan:
 - a. Laat interactief de bepaling van vector *HS* toe.
 - b. Voeg regels elf en twaalf samen.
 - c. Voeg een commentaarregel in die de instructies op regel dertien verklaart.
 - d. Bouw een controle in zodat de breedte van het histogram nooit groter is dan 80.
2. Bestudeer de hoofdfunctie voor tekstverwerking en breng de volgende veranderingen aan:
 - a. Laat het stoppen van invoer gebeuren door een éénletterige code.
 - b. Laat toe een code in te voeren die aangeeft dat ongeacht de uitvoer op formaat, er een nieuwe paragraaf moet worden begonnen.
3. De hulpfunctie voor het wijzigen van de tekst kan alleen maar een woord opzoeken. Breng een zodanige verandering aan dat bij het doorlopen van de tekst ook twee op elkaar volgende woorden kunnen worden nagekeken. Maak daarbij gebruik van het feit dat er in de op te zoeken vector een spatie kan voorkomen, en dat de positievector *POS* de positie van de spaties in de tekst weergeeft.
4. Bij het op formaat en met de rechter kantlijn afdrukken van een tekst zou het kunnen voorkomen dat het toevalsgetal *Z* altijd gelijk blijft. Dit zou betekenen dat de spaties tussen dezelfde woorden ingevoerd worden. Herprogrammeer de instructies zó dat *Z* nooit gelijk is aan zijn vorige waarde.
5. De functie voor budgettering in dit hoofdstuk is enigszins gebruikersvriendelijk. Er is echter geen foutencontrole geprogrammeerd. Breng deze controle alsnog aan zodat de gebruiker bij foutieve invoer de gegevens opnieuw kan invoeren. Doe dit voor de volgende interacties:
 - a. Limiteer het aantal te budgetteren maanden tot zes.
 - b. Controleer het maandnummer.
 - c. Controleer de gemaakte keuze uit het menu.
 - d. Controleer het nummer van de in te voeren maand.
 - e. Bij invoer van gegevens van één maand, controleer op zeer grote getallen.
 - f. Controleer de lengte van de vaste budgetvector bij invoer.
 - g. Controleer vóór het gebruik van het histogram of de post- of maandreferentie goed is en of de te tekenen waarden positief zijn.

6. Vervang de hulpfuncties voor het opstellen van vragen en antwoorden bij het COO voorbeeld door één enkele functie. Deze functie kan alsnog een tekst laten invoeren en vraagt naar een reeks van vraag/antwoord-combinaties. Gebruik eventueel deze nieuwe functie om de functie *LEER* te starten.
7. Bij de functie voor financiële planning is geen aandacht geschonken aan de volgorde van de instructies in het model. Weliswaar wordt een fout gemeld wanneer een rij- of kolomnaam verkeerd is, of nog niet is gedefiniëerd. Bij het afdrukken van het rapport komt men echter voor een probleem te staan wanneer de matrix *M* nog leeg is. Vang deze fout op via een bericht aan de gebruiker.
8. Schrijf een hulpfunctie die conversationeel werkt en toelaat een willekeurige functienaam met een financieel model, om te zetten naar de variabele *F*. Deze functie moet ook nakijken of er minstens één rij en één kolom zijn, en rapporteert het aantal van elk van de voorkomende instructiecodes. Deze functie kan eventueel de hoofdfunctie oproepen.
9. Ontwerp en programmeer een hoofdfunctie met eventueel een paar hulpfuncties waarmee een makelaar een aantal gegevens kan opgeven en later opvragen over diverse huizen die hij te koop heeft. Laat alles verlopen volgens een menu met nu en dan een vraag naar relevante parameters, en laat vriendelijke rapporten afdrukken. Zo bijvoorbeeld moet de functie, naast een mogelijkheid tot invoer, de volgende vragen kunnen beantwoorden, waarbij alles tussen de tekens $<$ en $>$ parameter-bepaald zijn:
 - a. Hoeveel huizen heb ik te koop?
 - b. Hoeveel huizen van twee-onder-één kap zijn er in het bestand?
 - c. Hoeveel huizen met een woonkameroppervlakte van meer dan $<40>$ vierkante meter zij er?
 - d. Hoeveel huizen zijn er in de gemeente $<ABC>$?
 - e. Zijn er huizen van exact $<140.000>$ gulden, en zo ja, welke?
 - f. Welke zijn de huizen tussen $<109.000>$ en $<129.000>$ gulden?
 - g. Hoeveel slaapkamers hebben de huizen boven $<200.000>$ gemiddeld?
 - h. Hoeveel vierkante meter totaal aan oppervlakte is er te koop in gemeente $<ABC>$?
 - i. enz.
10. Ontwerp en programmeer een interactieve functie die alle informatie bevat over het actief werkgeheugen zoals een beschrijving van de functies, hoe ze worden gebruikt, hoe de gegevens ervoor moeten worden aangebracht en hiervoor hulp aanbieden, en het laten uitvoeren van deze functies. Laat de gebruiker de detaillering van elke functie zelf bepalen. Eindig deze functie met het wegschrijven van alle gedane werk.

9 Organiseren van het werkgeheugen en het *APL*-systeem

De lezer heeft wellicht gemerkt dat de interactie tussen de programmeur en het *APL*-systeem via speciale opdrachten verloopt. Deze worden in *APL* gebruikt om met de computer te 'praten'. Dit gesprek verloopt via het *APL*-systeem dat de gegeven opdrachten vertaalt in machine-instructies. Hierdoor ontstaat een interactie tussen de gebruiker, het werkgeheugen en de computer. We maken een onderscheid tussen drie soorten van interacties. De eerste soort legt op organisatorisch gebied verband tussen het werkgeheugen en het *APL*-systeem. De tweede soort van interacties betreft de organisatie binnen het werkgeheugen zelf. De derde en laatste soort laat de gebruiker toe vanuit het werkgeheugen toegang te krijgen tot gegevens die niet in één of ander werkgeheugen zijn opgeslagen maar die via de interactie van het *APL*-systeem en het computersysteem wel bereikbaar zijn.

Tot nu toe hebben we *APL* uitsluitend binnen het actief werkgeheugen toegepast. Hierbij wordt soms vergeten dat men met een systeem bezig is waarvan de kracht niet alleen in de taal zelf ligt om allerlei moeilijke problemen op een gemakkelijke manier op te lossen, maar waarbij men ook de mogelijkheid heeft met de computer zélf te communiceren. Eén van de voornaamste mogelijkheden is bijvoorbeeld dat via het *APL*-systeem gegevens kunnen worden opgevraagd uit grote gegevensbanken en dat deze gegevens op een geprogrammeerde manier kunnen worden verwerkt. Dit laatste noemen we de data-communicatie van *APL*. Om dit in de juiste context te plaatsen, bespreken we eerst de communicatie tussen de gebruiker en het systeem vanuit het werkgeheugen zelf.

9.1 Het *APL*-systeem en het werkgeheugen

9.1.1 Manipulatie van het werkgeheugen

Een werkgeheugen kan men voorstellen als een groot schrijfbord waarop men berekeningen en aantekeningen maakt. Men kan dat bord ook ergens wegzetten en een nieuw bord gaan beschrijven. Ook dit nieuwe bord kan voor verder gebruik ergens geplaatst worden. Zolang de voorraad borden strekt kan men doorgaan. In *APL* gebruikt men een werkgeheugen op dezelfde manier. Daarbij wordt het bord waarop men op een bepaald moment aan het schrijven is, het *actief werkgeheugen* genoemd. Dit werkgeheugen kan worden weggeschreven naar een *bibliotheek* van

werkgeheugens. Dit hebben we reeds gezien bij gebruik van het commando *)SAVE*. Het heeft echter geen zin een werkgeheugen weg te schrijven zonder dat deze een naam heeft gekregen. Deze naamgeving gebeurt met het commando *)WSID* zoals in:

```
)WSID OEFEN
WAS CLEAR WS
```

Het systeem geeft onmiddellijk antwoord door het afdrukken van de vorige naam van het werkgeheugen. Indien *WAS CLEAR WS* verschijnt, betekent dit dat het werkgeheugen nog geen naam bezit. Wanneer men niet meer weet wat de naam van een actief werkgeheugen is kan met dit met hetzelfde commando opvragen, echter zonder toewijzing van een naam:

```
)WSID
OEFEN
```

Ongeacht het feit of een werkgeheugen een naam bezit of niet, blijft alles wat we binnen dit geheugen doen of bepalen voor dit werkgeheugen geldig, ook al wordt deze weggeschreven en later weer als actief werkgeheugen gebruikt. Men kan immers alleen maar in het actief werkgeheugen veranderingen en toevoegingen aanbrengen. Een reeds benoemd en weggeschreven werkgeheugen maakt men actief door het commando *)LOAD* te gebruiken:

```
)LOAD TEST
SAVED 13:02:00 21-10-81
```

Het systeem antwoordt nu met enige informatie over het werkgeheugen in de vorm van de datum en de tijd wanneer het werkgeheugen het laatst werd weggeschreven. Probeert men echter een werkgeheugen actief te maken dat niet in de bibliotheek van werkgeheugens van de gebruiker staat, dan krijgt men een andere melding:

```
)LOAD TESTQ
WS NOT FOUND
)LOAD TESTQ
^
```

Om na te gaan wat er in de bibliotheek aanwezig is kan men het commando *)LIB* gebruiken. Als er werkgeheugens zijn weggeschreven, verschijnt er een lijst van de namen, inclusief de naam van het werkgeheugen dat eventueel actief is:

```
)LIB
APLWERK1
INLEIDING
OEFEN
TEST
```


Een nieuw werkgeheugen kunnen we met het *)SAVE*-commando op twee verschillende manieren wegschrijven. Ofwel men geeft het werkgeheugen meteen een naam, en schrijven deze dan weg naar de bibliotheek:

```

)WSID CONTACT
WAS CLEAR WS
)SAVE
13:02:00 21-10-81 CONTACT
    
```

ofwel men geeft de naam bij het commando zelf mee. Eventueel mag een dubbele punt en een wachtwoord na de naam worden opgegeven:

```

)WSID
CLEAR WS
)SAVE CONTACT
13:02:00 21-10-81 CONTACT
    
```

Het wegschrijven van een werkgeheugen kan echter ook nog op een paar andere manieren. Bij verbreking van de verbinding tussen de terminal en de computer of bij het uitvallen van de computer, wordt het actief werkgeheugen meestal automatisch weggeschreven onder de naam *CONTINUE*. Bij de eerstvolgende verbinding komt er een melding dat het gedane werk onder die naam is weggeschreven en dat dit meteen ook de naam van het actief werkgeheugen is. Door een eventuele naamsverandering kan men deze onderbreking herstellen. We maken er op attent dat deze procedure afhankelijk kan zijn van het *APL*-systeem waarmee men werkt. Heeft deze onderbreking plaatsgevonden tijdens de uitvoering van een functie, dan doet men er, ongeacht het type systeem, goed aan de statuslijst na te kijken met *)SI*.

Het is mogelijk het actief werkgeheugen weg te schrijven met de naam *CONTINUE*. Men gebruikt hiervoor het volgende commando:

```

)SAVE CONTINUE
13:02:00 21-10-81 CONTINUE
    
```

Hierbij verloopt het wegzetten van het werkgeheugen net zoals bij een werkgeheugen van een andere naam. Het is ook mogelijk het werkgeheugen op zo'n manier weg te schrijven dat de verbinding met de computer verbroken wordt en dat bij de eerstvolgende aansluiting het werkgeheugen automatisch als actief geheugen kan worden gebruikt. Dit kan alleen met het volgende commando:

```

)CONTINUE
014: 13:0:00 21-10-81
CONNECTED 0:01:10 CPU 0:00:02
  A NIEUWE AANSLUITING
SAVED 13:02:00 21-10-81 CONTINUE
    
```

Veronderstellen we even dat er een hulpfunctie is weggeschreven in het werkgeheugen *OEFEN*. We bevinden ons nu in een ander actief werkgeheugen en willen de functie ook hierin beschikbaar hebben. Dit kunnen we doen door het kopiëren van de functie uit het niet-actief werkgeheugen. Als we hierbij vergeten het wachtwoord op te geven, antwoordt het systeem met het duidelijke bericht:

```
)COPY OEFEN ALS
WS LOCKED
)COPY OEFEN ALS
^
NOT COPIED: ALS
```

Bij bovenstaand *COPY*-commando wordt een functie, een variabele of een groep functies gekopieerd, ongeacht de aanwezigheid van gelijknamige functies of variabelen in het actief werkgeheugen. Dit wil zeggen dat in die gevallen waar deze functies of variabelen reeds in het actief werkgeheugen voorkomen, een eventuele andere versie uit het ander werkgeheugen hun plaats inneemt. Om dit euvel te vermijden, doet men er meestal goed aan een vorm van 'beschermend' kopiëren te gebruiken, zoals in:

```
NAAM←'JAN'
)PCOPY OEFEN ALS FOUT TEKST NAAM
SAVED 13:02:00 21-10-81
NOT FOUND: TEKST
NOT COPIED: NAAM
```

Hierbij is de variabele *NAAM* niet gekopieerd en is de waarde ervan niet veranderd. De variabele *TEKST* kan niet worden gekopieerd omdat deze niet in het werkgeheugen *OEFEN* voorkomt.

Als men het werkgeheugen volledig wil uitwissen, net zoals bij een bord, kan men dit doen met:

```
)CLEAR
CLEAR WS
```

Het resultaat is dan dat niet alleen alle functies, variabelen of groepen van functies verwijderd zijn, maar dat alle systeemvariabelen op hun oorspronkelijke waarde worden gezet en dat het werkgeheugen geen naam meer heeft.

Het gebeurt ook wel eens dat een werkgeheugen in de bibliotheek van de gebruiker niet meer van belang is. Men kan deze dan verwijderen met het commando:

```
)DROP CONTAC
13:02:00 21-10-81
```


Men doet er altijd goed aan even op te letten of men wel de juiste naam van het weggeschreven werkgeheugen heeft, en of er inderdaad geen gegevens of functies instaan die later nog van pas zouden kunnen komen. Maakt men die fout toch dan is het in de meeste gevallen mogelijk in een later stadium het werkgeheugen, via de beheerder van het APL-systeem, alsnog terug te krijgen.

9.1.2 Communicatie met andere gebruikers

Vanuit het actief werkgeheugen is het mogelijk enige communicatie te onderhouden met andere gebruikers die met het APL-systeem verbonden zijn. Natuurlijk is dit alleen van toepassing in die gevallen waar men meerdere gebruikers heeft. Bepaalde microcomputers hebben niet de mogelijkheid APL simultaan te gebruiken. Bij grotere systemen is het vaak mogelijk van gebruiker tot gebruiker berichten door te sturen. De exacte wijze waarop dit gebeurt is echter afhankelijk van het APL-systeem dat wordt gebruikt. We verwijzen de lezer daarom naar de beschrijving beschikbaar gesteld door de leverancier. Dit geldt voor vele van de onderstaande commando's.

Zo kan men te weten komen hoeveel gebruikers er zijn en wie deze gebruikers zijn. Dit laatste is soms moeilijk te bepalen aangezien alleen de eerste drie letters van een naam of code worden afgedrukt. Dit blijkt uit onderstaand voorbeeld:

```
      )PORTS
001: OPR
007: MAP
020: UYT
021: ADR
```

De lijst van aansluitingen bestaat uit een nummer dat we het poortnummer noemen en de code of afkorting van de naam van de persoon die via deze poort op het APL-systeem is aangesloten. Soms ziet men in deze lijst dat de computer-operator aangesloten is onder de naam OPR. Men kan naar deze laatste door middel van het volgende commando, een bericht sturen of een vraag stellen:

```
      )OPRN HALLO, ALLES WERKT PRIMA
```

waarna men weer gewoon verder kan werken. De letter N betekent dat er niet noodzakelijk een antwoord wordt verwacht. Wil men echter op een antwoord wachten dan kan men het commando)OPR gebruiken. Als men vindt dat dit te lang duurt kan alsnog de ATTENTION-toets worden gebruikt zodat men verder kan werken.

Op een bijna identieke manier kan men berichten naar andere gebruikers sturen. Dit doen we door het commando)MSG te gebruiken, gevolgd door het poort-

nummer dat voor de naamafkorting of de code staat, en het bericht. Bij de andere gebruiker komt het bericht binnen in de vorm van het aansluitnummer dat het bericht verstuurd heeft, gevolgd door een dubbele punt en de tekst. Staat echter na de dubbele punt ook nog eens de letter *R* onderlijnd, dan betekent dit dat de gebruiker een antwoord verwacht. Dit zien we in de volgende interactie:

```

)MSG 14 HALLO JAN
014: R OOK DAG - WAT WIL JE?
)MSG 14 HOE VERANDER IK DE NAAM VAN DEZE WS?
014: HEEL EENVOUDIG: )WSID <NAAM>
)MSGN 14 DANK JE EN TOT ZIENS
014: DAAAAAG

```

Bij de meeste systemen kan een bericht alleen maar worden ontvangen als de gebruiker actief met iets bezig is. Tussen twee instructies in kan dan een bericht doorkomen. Zit men rustig voor de terminal zonder dat er een functie uitgevoerd wordt, dan komt geen bericht binnen. Verwacht men echter een bericht, dan kan de gebruiker zich actief tonen door een bericht met vraag naar antwoord naar zichzelf te sturen. Als aansluitingsnummer mag dan nul of het eigen nummer worden gebruikt:

```

)MSGN 0 IK BEN ACTIEF
020: IK BEN ACTIEF

```

Bij bepaalde APL-systemen is het zelfs mogelijk alle vormen van berichtgeving van andere gebruikers af te weren. Dit doet men door het commando *)MSG OFF* in te typen. Op dat moment komen echter ook geen berichten binnen die van algemeen belang zijn en door *PA*: worden gekenmerkt.

Wil men weer berichten ontvangen dan gebruikt men *)MSG ON* waarna de laatste algemene boodschap alsnog wordt afgedrukt.

9.1.3 Publieke bibliotheken

Na verloop van tijd zal de APL-gebruiker enkele werkgeheugens weggeschreven hebben in de eigen bibliotheek. Daar kan verder niemand inkomen zonder het aansluitingsnummer, de naam van het werkgeheugen en eventueel het wachtwoord op te geven. Bij de grote computers staat over het algemeen een hele reeks werkgeheugens ter beschikking die een grote hoeveelheid algemene functies en hulpfuncties bevat. Deze werkgeheugens zijn gerangschikt in genummerde bibliotheken. Als men het nummer van zo'n bibliotheek weet dan kan men een lijst vragen van al de daarin aanwezige functies door als volgt een commando in te typen:


```
)LIB 99
AUTO
PLOT
STATISTICS
VARIANCE
WSLIST
```

Wil men uit deze bibliotheek 99 het werkgeheugen *AUTO* kopiëren dan doet men dit als volgt:

```
)COPY 99 AUTO
SAVED 09:00:01 14-03-74
```

Op een zelfde manier is het mogelijk bij bepaalde *APL*-systemen op grote computers, een zelfgemaakt werkgeheugen weg te schrijven als:

```
)SAVE 99 OEFEN
13:02:00 21-10-81 OEFEN
```

Om alleen maar functies uit een werkgeheugen in een bibliotheek naar het eigen actief werkgeheugen te brengen kunnen we deze kopiëren met:

```
)COPY 99 AUTO STUUR
SAVED 09:00:01 14-03:74
COPIED: STUUR
```

Hierbij gelden dezelfde regels als bij het kopiëren vanuit een eigen bibliotheek, maar het is soms beter het commando *)COPY* te gebruiken.

9.1.4 Aansluiten op een *APL*-systeem

Het aansluiten met een nummer op een *APL*-systeem kan per systeem verschillen. We geven hier een algemene vorm aan die eventueel moet worden voorafgegaan door een andere vorm van interactie met het computersysteem. De meest gebruikte vorm is deze:

```
)12345678
020: 13:02:00 21-10-81
APL-SYSTEEM XXX
|
```

Hierbij wordt het poortnummer, de tijd van aansluiting, de datum en de naam van de gebruiker afgedrukt. Deze regel wordt meestal gevolgd door een benaming van het *APL*-systeem waarmee men werkt. Indien deze aansluiting volgt na een

onderbreking van de computer, zal wellicht worden gemeld dat alle informatie tot op dat moment in het werkgeheugen *CONTINUE* is opgeslagen.

9.1.5 Afsluiten van de interactie

Het beëindigen van een interactie met het *APL*-systeem en de computer kan men op twee manieren doen. Of men verbreekt de verbinding met zowel *APL* als de computer, of men blijft nog met het systeem verbonden via het netwerk van kabels of telefoonlijnen. In het eerste geval volstaat het commando *)OFF* waarna een melding komt die per systeem varieert. In de meeste gevallen wordt de tijd van afsluiting meegegeven, de datum, het aantal verbruikte rekeneenheden enz. In het tweede geval gebruikt men *)OFF HOLD* waarna men opnieuw een aansluiting kan vragen. Ook dan verschijnt eerst enige informatie over het gebruik. Zoals we reeds hebben gezien heeft het gebruik van de *)CONTINUE* een afsluiting tot gevolg.

Bij deze commando's is het toegestaan een wachtwoord mee te geven door het commando te vervolgen met een dubbele punt en het wachtwoord. Dit wachtwoord geldt alleen voor het aansluitingsnummer, dus niet voor een werkgeheugen. Men doet er toch goed aan dit ergens te noteren, want het aansluitingsnummer is zonder dit wachtwoord niet meer te gebruiken.

Bij sommige *APL*-systemen is het mogelijk af te sluiten zonder eerst *)OFF* te gebruiken. Men kan immers opnieuw aansluiten door het invoeren van het nummer met eventueel een wachtwoord. Ook dan verschijnt een melding van tijd, datum, enz., zoals bij een normale afsluiting. Deze melding wordt gevolgd door informatie over deze nieuwe aansluiting.

9.2 Organisatie van het werkgeheugen

Net zoals bij de interactie van werkgeheugen en *APL*-systeem kan men systeemcommando's gebruiken die binnen een bepaald werkgeheugen van toepassing zijn. Onder interactie is hier te verstaan de communicatie tussen de gebruiker, met bepaalde wensen voor het werkgeheugen, en het *APL*-systeem. Om deze wensen kenbaar te maken, kan men gebruik maken van systeemcommando's, systeemvariabelen en systeemfuncties. De eerste groep wordt gekenmerkt door het gebruik van een openend haakje (*()*) en een beschrijvend trefwoord, terwijl de tweede en de derde groep gevormd worden door een rechthoekje (*[]*) gevolgd door een tweeletterige afkorting van de variabele of functie. Het onderscheid tussen de laatste twee wordt impliciet door de naam gegeven. De variabele bevat namelijk een constante die een rol speelt bij het gebruik van *APL* in het werkgeheugen, terwijl de systeemfuncties een duidelijke functie vervullen, en dus een executie tot gevolg hebben. Hierbij zij opgemerkt dat niet op alle *APL*-systemen de systeemvariabelen en -functies zo uitgebreid voorkomen.

9.2.1 Systeemcommando's

Het commando *)DIGITS* werd reeds uitvoerig in hoofdstuk 7 besproken. Het komt overeen met de systeemvariabele $\square PP$, en beide hebben tot taak het aantal significante cijfers voor de uitvoer te bepalen. De significantie voor berekeningen is afhankelijk van het gebruikte *APL*-systeem. Dit wordt over het algemeen uitgedrukt in het aantal cijfers voor en na de decimale punt, meestal tien.

Na het werken in een werkgeheugen zijn er meestal enkele variabelen en functies gedefinieerd die daarna niet meer nodig zijn. Deze kan men uit het werkgeheugen verwijderen door het commando:

)ERASE V,X,TESTO

Soms volgt een lijst van die variabelen of functies die inderdaad zijn uitgewist. Heeft men per ongeluk een verkeerde functie of variabele verwijderd, dan kan men daarvan misschien nog een kopie in een werkgeheugen in de bibliotheek halen. Als voorwaarde geldt natuurlijk dat een werkgeheugen met de gewenste kopie is weggeschreven.

Voordat men een wijziging in het aantal variabelen of functies in een werkgeheugen aanbrengt, is het goed regelmatig na te kijken wat zoal is gedefinieerd. Hiervoor zijn twee commando's beschikbaar, namelijk:

```

)FNS
ALS   BUDGET   DRUKT   FINAN   FOUT   INIT
MODEL WIJZIG   Δ
)VARS
F     BUDGM    NO      NV      TEKST

```

Soms is het nuttig enkele van deze functies of variabelen te groeperen. Dit kan men bereiken met het commando *)GROUP*, gevolgd door de naam van de groep en de functies of variabelen in deze groep zoals in:

)GROUP FP ALS FOUT FINAN MODEL Δ M

Twee verwante commando's worden bij de groepering gebruikt. Het eerste geeft aan wat de groepsnamen zijn, en het tweede geeft die functies of variabelen aan die zich in de groep bevinden:

```

)GRPS
FP
)GRP FP
ALS   FOUT   FINAN   MODEL   Δ   M

```

De groepsnamen kunnen zelf als variabelenamen worden gebruikt zodat ze ook weer verwijderd kunnen worden. Het verwijderen van de naam van de groep heeft echter niet tot gevolg dat de variabelenamen of functie namen ook verwijderd zijn, zoals blijkt uit:

```

) VARS
BUDGM F M NO NV TEKST
) ERASE FP
) FNS
ALS BUDGET DRUKT FINAN FOUT INIT
) GRPS

```

Zoals men kan zien, is de naam van de groep niet in de lijst van variabelen terug te vinden. Indien een functie in een werkgeheugen wordt gewist, en deze functie wordt nog vermeld in de statuslijst, dan volgt hierop een melding. Bijvoorbeeld:

```

) SI
BUDGET[3] *
FINAN[16] *
) ERASE BUDGET
SI DAMAGE
) SI
*
FINAN[16] *

```

Het commando `)SI` dient zoals reeds eerder opgemerkt om een lijst te verkrijgen. Een daaraan verwant commando `)SIV` geeft niet alleen aan op welke regel in een functie de onderbreking heeft plaatsgevonden, maar ook wat de openstaande lokale variabelen voor deze functie zijn, zoals bijvoorbeeld in:

```

) SI
BUDGET[3] *
) SIV
BUDGET[3] * K I HS VST MND
PERΔPOST BAL NEW BER PAP HIS
PP END

```

Het verwijderen van variabelen en functies is soms noodzakelijk omdat bij de definitie van een functie of de specificering van een variabele de volgende melding wordt gegeven:

```

A←1
SYMBOL TABLE FULL
A←1
^

```


Het beschikbaar geheugen voor namen is opgebruikt. Men kan dan proberen een paar niet-gebruikte variabelen te verwijderen, of men kan de tabel die de symbolen (namen) bevat vergroten. Dit doet men met het commando:

```
      )SYMBOLS 300
WAS 256
```

Dit commando kan echter alleen maar in een leeg werkgeheugen worden uitgevoerd. Het getal dat volgt bepaalt de grootte van de tabel. Men kan dan als volgt te werk gaan. Schrijf het actief werkgeheugen weg, maak een leeg werkgeheugen, verander de grootte van de tabel en kopieer het weggeschreven werkgeheugen. Het actief en naamloos werkgeheugen krijgt daarna de naam van het gewenste werkgeheugen en wordt weggeschreven. Dit gebeurt als volgt:

```
      )SAVE
13:02:00 21-10-81 OEFEN
      )CLEAR
CLEAR WS
      )SYMBOLS 400
WAS 256
      )COPY OEFEN
SAVED 13:02:00 21-10-81
      )WSID OEFEN
WAS CLEAR
      )SAVE
13:02:00 21-10-81 OEFEN
```

Het aantal symbolen in een tabel wordt voor de meeste systemen zonder deze ingreep vastgesteld op 256. Voor microcomputers kan dit echter een kleiner getal zijn. De minimum en maximum grootte van de tabel zijn bij de meeste systemen respectievelijk 26 en 4241.

Het systeemcommando `)WIDTH` wordt gebruikt om de breedte van een uitvoerregel te bepalen. Voor een beeldscherm terminal is dit meestal 80, terwijl men op de meeste afdrukeenheden 130 posities kwijt kan. Dit commando is gelijk aan de systeemvariabele `□PW` en heeft in principe de waarde 130.

Als laatste systeemcommando bespreken we de beginwaarde voor de indexgeneratie. Dit commando is gelijk aan de systeemvariabele `□IO` en heeft in principe altijd de waarde één. Bij verandering naar de waarde nul begint de index op nul. Dit betekent dat de index van een vector ook nul mag zijn zoals hieronder wordt getoond:

```

      □IO
1      V←1 2 3 4 5
      V[□IO]
1
      □IO←0
      V[□←15]
0 1 2 3 4
1 2 3 4 5

```

Het effect bij enkele primitieve functies is als volgt:

```

      □IO←0
      5?5
2 0 4 3 1
      10÷12
DOMAIN ERROR
      10÷12
      ^
      !15
1 1 2 6 24
      ~11
1

```

9.2.2 Systeemvariabelen

De meeste APL-systemen beschikken over een dertiental systeemvariabelen die binnen een actief werkgeheugen door de gebruiker kunnen worden aangepast. Het zijn de volgende systeemvariabelen:

```

□IO
□PP
□PW
□AI
□AV
□CT
□LC
□LX
□RL
□TT
□TS
□UL
□WA

```


waarbij de eerste drie reeds werden besproken als systeemcommando's. Deze zijn respectievelijk *)ORIGIN*, *)WIDTH*, en *)DIGITS*.

De systeemvariabele $\square AI$ wordt gebruikt om informatie over de aansluiting te verkrijgen. Deze informatie bevat het aansluitingsnummer van de gebruiker, de hoeveelheid gebruikte rekentijd sinds de aansluiting, de duur van de aansluiting, en de tijd dat het systeem heeft gewacht op invoer of instructies. Bijvoorbeeld:

$\square AI$
12345678 4321 6534 8943

De laatste drie hoeveelheden zijn uitgedrukt in seconden.

Reeds in het eerste hoofdstuk hebben we de systeemvariabele $\square AV$ gezien. Deze variabele is een vector die meestal uit 256 karakters, tekens, samengestelde tekens en controles voor het toetsenbord bestaat. Deze vector wordt de *atomic vector* genoemd. Elk element in deze vector mag worden gebruikt en mag via indicering aan lokale of globale variabelen worden toegewezen. Deze vector hoeft niet bij alle *APL*-systemen gelijk te zijn.

De exactheid van de voorstelling van getallen in de computer laat soms wel eens te wensen over, vooral bij het berekenen van zeer grote of zeer kleine getallen. Vooral bij delingen gebeurt het wel eens dat door de binaire omzetting in de computer, het resultaat van de deling van twee getallen zoals $2 \div 4$, niet exact blijft behouden. Het kan dus voorkomen dat het resultaat intern als 2.000000000001 wordt weergegeven, en dat men bij vergelijking met de waarde 2 op een ongelijkheid stuit. Een systeemvariabele geeft de mogelijkheid te controleren hoeveel cijfers bekeken moeten worden om twee getallen aan elkaar gelijk te verklaren. Wat dus moet worden bepaald is de tolerantie van de vergelijking. De variabele $\square CT$ bevat een zeer kleine constante, in principe $1E^{-13}$, die intern wordt gebruikt bij de vergelijking van twee getallen. Bij deze vergelijking is bijvoorbeeld het getal 13 genomen als aanduiding van het aantal cijfers dat moet worden vergeleken. Als twee getallen tot het dertiende cijfer gelijk zijn dan wordt bij een logische vergelijking het resultaat één gegeven, anders nul. Een paar voorbeelden tonen dit. We nemen daarbij voor de tolerantie telkens andere waarden:

```

A1←2.0001
A2←2.00001
A3←2.000001
□CT←1E-10
A1=A2
0
A2=A3
0
```

```

                                □CT←1E-4
                                A1=A2
0
                                A2=A3
1
                                □CT←1E-2
                                A1=A2=A3
1

```

De systeemvariabele $\square LC$ bevat net zoals de statuslijst informatie over de regelnummers waar een functie of hulpfunctie werd onderbroken. Het verschil met $)SI$ is dat dit alleen voor de laatst uitgevoerde functie geldt. Men kan dus om een functie te hervatten, in plaats van de verwijzing $\rightarrow$ en het regelnummer te gebruiken, ook $\rightarrow \square LC$ invoeren. In de gevallen waar meerdere hulpfuncties werden gebruikt, bevat de vector de regelnummers in volgorde van de laatst uitgevoerde (hulp)functie tot hoofdfunctie. Een paar voorbeelden illustreren het gebruik van deze systeemvariabele:

```

                                DRUKT
                                KIES UIT:
                                1 CONTINUE AFDRUK
                                2 FORMAAT AFDRUK
                                3 STOP AFDRUKKEN

```

```

                                )SI
                                DRUKT[5] *
                                □LC
                                5
                                →□LC
                                3

```

De meeste APL-systemen bieden ook de mogelijkheid een latente instructie in het werkgeheugen in te bouwen. De bedoeling hiervan is dat bij het activeren van het werkgeheugen deze latente instructie meteen wordt uitgevoerd. Voor men het werkgeheugen wegschrijft kan men de systeemvariabele $\square LX$ specificeren met een commando, een berekening, een functienaam, een verwijzing tot continuering van een onderbroken functie, enz. Deze toewijzing heeft de vorm van een karaktervector:

```

                                □LX←''' VEEL SUCCES '''

```

Schrijven we het werkgeheugen weg, om het dan later opnieuw actief te maken, dan gebeurt er automatisch het volgende:


```

)SAVE
13:02:00 21-10-81 OEFEN
)LOAD OEFEN
SAVED 13:02:00 21-10-81
VEEL SUCCES

```

□*LX* is een karaktervector die in feite gebruikt wordt als □*LX*. Omdat de informatie in de variabele een vector is die bij het activeren van het werkgeheugen alleen maar dient te worden afgedrukt, moet de informatie binnen de karaktervector zelf tussen aanhalingstekens staan. Aangezien het gebruik van een aanhalingsteken in een tekst, die op zich zelf ook al tussen aanhalingstekens staat, moet worden verdubbeld, zijn er bij het hierboven getoonde voorbeeld inderdaad drie aanhalingstekens nodig. Anders is het bij de opdracht een functie uit te voeren, of bij het hervatten van een functie. Men specificeert de variabele dan als:

```

□LX←'→□LC'
□LC←'FINAN F'

```

De volgende systeemvariabele die we bespreken is zeer relevant voor diegenen die met willekeurige getallen gaan werken. Het is namelijk zo dat men bij het activeren van het werkgeheugen altijd met hetzelfde willekeurig zou beginnen omdat dit gebaseerd is op een getal dat vast staat. Dit laatste getal, in het Engels *seed* genoemd, wordt bij verder gebruik in het werkgeheugen altijd veranderd. Men kan na activering onmiddellijk 25 invoeren, waarbij een bepaald getal tussen nul en zes wordt gegeven, bijvoorbeeld drie. Een volgende keer bijvoorbeeld 1, enz.. Activeert men dit werkgeheugen echter opnieuw, dan zal bij dezelfde instructie dezelfde resultaten worden gekregen, omdat het getal dat de *seed* voorstelt weer op zijn oorspronkelijke waarde is gezet. Men kan met de systeemvariabele □*RL* deze waarde veranderen door een groot getal te specificeren dat tussen 1 en $-1+2 \times 31$ ligt. Bijvoorbeeld:

```

)CLEAR
CLEAR WS
?10
4
)CLEAR
CLEAR WS
□RL←101
?10
7

```

De systeemvariabele □*TT* wordt zelden gebruikt en dient om het type terminal dat in gebruik is op te vragen. Als antwoord verschijnt dan een bepaalde code (0, 1, 2 of 3), waarvan de betekenis van het gebruikte *APL*-systeem afhangt. De pro-

grammeur kan eventueel en afhankelijk van het type terminal de uitvoerformaten aanpassen.

De systeemvariabele $\square TS$ is de *tijdvariabele* die de tijd aangeeft. Als uitvoer verschijnt een vector met zeven elementen, waarbij het eerste element het jaartal is, het tweede de maand, gevolgd door de dag, het uur, de minuut, de seconde en als laatste de milliseconden. Bij rapportering kan deze informatie handig worden gebruikt zoals in hoofdstuk 8 in het financiële analyse-voorbeeld.

De systeemvariable $\square UL$ dient om aan te geven hoeveel gebruikers er op een bepaald moment zijn. Het geeft dus niet aan wie er aangesloten zijn. Hiervoor kan men het commando $\rangle PORTS$ gebruiken.

Een laatste systeemvariabele is $\square WA$. Deze belangrijke variabele geeft aan hoeveel bytes er in het actief werkgeheugen nog vrij zijn. De grootte van een actief werkgeheugen is zeer sterk afhankelijk van het APL-systeem.

Als er reeds enkele functies zijn geschreven en tientallen variabelen met gegevens in het werkgeheugen zijn aangebracht, is het soms belangrijk te weten of er nog genoeg geheugen is om nog meer gegevens in te voeren. Hierbij kan men als vuistregel hanteren dat doorgaans één karakter één byte inneemt, dat een logische 0 of 1, één bit inneemt, en dat andere getallen vier tot acht bytes innemen.

9.2.3 Systeemfuncties

De systeemfuncties voor de interactie binnen het werkgeheugen zijn te interpreteren als gedefinieerde functies die voor de gebruiker werden geschreven, omdat binnen de APL-taal niet de mogelijkheid bestaat op een bepaalde manier variabelen en andere gedefinieerde functies te behandelen. De functienamen beginnen net zoals bij de systeemvariabelen met het *quad* teken $\square$, en zijn of monadisch of dyadisch. De systeemfunctie $\square CR$ hebben we reeds toegepast in het vorige hoofdstuk en dient om een gedefinieerde functie in een karaktermatrix om te zetten, zoals in:

$$F \leftarrow \square CR \ ' \Delta'$$

De variabele F ziet er als volgt uit:

$$\begin{array}{l} Z \leftarrow LINKS \ \underline{\Delta} \ RECHTS \\ Z \leftarrow LINKS \end{array}$$

We zien dus dat de regelnummers en het teken ∇ zijn weggefallen. We mogen echter niet vergeten dat de functienaam met eventueel de lijst van lokale variabelen op de eerste rij van de matrix staat. De oorspronkelijke functie is in het werkgeheugen gebleven en in feite hebben we dus een kopie gemaakt. Als we niet

de naam van een onverzegelde functie opgeven, is het resultaat een matrix met dimensies 0 0.

De tweede systeemfunctie $\square FX$ voert de omgekeerde bewerking uit, dus van een matrix naar een functie. Het gegeven moet een karaktervector zijn zoals in:

```

      ANDER
VALUE ERROR
      ANDER
      ^
      F← $\square CR$  'ANTWOORD'
      ρF
7 43
      F[1;]←43↑'ANDER'
ANDER
      ∇ANDER
[7] ∇

```

De systeemfunctie kan echter geen nieuwe functie maken indien de naam reeds bestaat als de naam van een functie die werd onderbroken, of als naam van een variabele, groep of regel in het werkgeheugen. Bij het omzetten wordt dan het regelnummer waar de fout ontstond als resultaat weergegeven. De meeste toepassingen van de functies $\square CR$ en $\square FX$ zijn te vinden bij het verwijderen of invoegen van regels in een functie.

Bij deze beide systeemfuncties werd de oorspronkelijke vorm altijd bewaard. Men kan echter variabelen en functies in het werkgeheugen verwijderen met het commando *ERASE*. De variabelen zijn altijd globaal omdat de lokale variabelen niet kunnen worden opgevraagd nadat de uitvoering van een functie is beëindigd. De mogelijkheid bestaat nu om de lokale variabelen en een eventueel gedefinieerde functie gedurende de uitvoering te verwijderen. Dit noemen we het *dynamisch* verwijderen. De systeemfunctie $\square EX$ wordt hiervoor gebruikt, bijvoorbeeld:

```

      ∇VERWIJDER;I
[1] I←?10
[2] I÷10
[3] J← $\square EX$  'I'
[4] →1
      ∇
      SΔVERWIJDER←4
      VERWIJDER
.3
VERWIJDER[4]
      I
VALUE ERROR
      I
      ^

```

Een vierde systeemfunctie laat toe een matrix te bouwen met de namen van de gebruikte namen van regels, variabelen of een gedefinieerde functie. Hiertoe gebruiken we $\square NL$ in de monadische vorm, waarbij als invoer de code 1,2 of 3 voor de respectievelijke namen wordt gegeven:

```

)FNS
ALS  BUDGET  INIT  FOUT  Δ
)VARs
BUDGM  MAANDEN  NM      NO      POSTEN
 $\square NL$  1

 $\square NL$  2
BUDGM      .
MAANDEN
NM
NO
POSTEN
 $\square NL$  3
ALS
BUDGET
INIT
FOUT
Δ

```

De namen van regels worden alleen van die functies gegeven die onderbroken zijn. Wenst men een aantal namen te weten die met een bepaalde letter beginnen, dan kan men de systeemfunctie in de dyadische vorm gebruiken als:

```

'A'  $\square NL$  3
ALS
'Z'  $\square NL$  2

```

In die gevallen dat er geen namen van regels, variabelen of functies in het werkgeheugen aanwezig zijn, is het resultaat van de executie van $\square NL$ een lege matrix met de dimensies 0 0.

De systeemfunctie $\square NC$ houdt verband met de functie $\square NL$. Men gebruikt deze functie immers om te weten te komen of een gegeven naam of een matrix van namen, correspondeert met regels, variabelen of functies. De functie genereert een vector met dezelfde lengte als het aantal namen. De waarden in de vector komen in volgorde overeen met de rij van namen. Er zijn vier mogelijke waarden:

- 0 De naam komt niet voor in het werkgeheugen.
- 1 De naam is die van een regel.
- 2 De naam is die van een variabele.
- 3 De naam is die van een functie.

Bijvoorbeeld:

```

           $\nabla FN$ 
[1]       $I \leftarrow 0$ 
[2]       $HIER: \rightarrow 0 \text{ ALS } 3 < I \leftarrow I + 1$ 
[3]       $\rightarrow HIER \triangle \square \leftarrow I * 2$ 
           $\nabla$ 
           $FN$ 

1
4
       $\square_{WC} 'I'$ 

2
       $\square_{WC} 'HIER'$ 

0
       $\square_{WC} 'FN'$ 

3

```

De laatste systeemfunctie $\square_{DL}$ laat toe het systeem in een wachttoestand te brengen. De functie is dus een *delay* waarbij in de monadische vorm het aantal seconden wachttijd wordt opgegeven. De functie heeft als uitvoer het eigenlijke aantal seconden dat er is gewacht omdat hiervan slechts een benadering kan worden verwacht. We kunnen bijvoorbeeld het systeem tien seconden laten wachten door de functie te gebruiken als:

```

       $\square_{DL} 10$ 

10

```

9.2.4 Systeem-afhankelijke functies

Niet bij alle *APL*-systemen kan men gebruik maken van systeemvariabelen of systeemfuncties. Bij het initieel ontwerp van *APL* werd ervoor gezorgd dat de interactie met het werkgeheugen mogelijk zou zijn via de **I**-functie. Het monadisch gebruik van deze functie laat toe enkele van de voornaamste systeemvariabelen te imiteren. Het nummer dat volgt op het teken **I** geeft aan wat de systeemvariabele is. Hieronder volgt een lijst van mogelijke functies **I**. Waar mogelijk is gerefereerd naar de systeemvariabele die hetzelfde resultaat geeft:

```

I19   $\leftrightarrow \square_{AI}[5]$ 
I20   $\leftrightarrow \square_{TS}[4 \ 5 \ 6]$ 
I21   $\leftrightarrow \square_{AI}[4]$ 
I22   $\leftrightarrow \square_{WA}$ 
I23   $\leftrightarrow \square_{UL}$ 
I25   $\leftrightarrow \square_{TS}[13]$ 
I26   $\leftrightarrow \square_{LC}[1]$ 
I27   $\leftrightarrow \square_{LC}$ 

```

9.3 Interactie met het computersysteem (*shared variables*)

Enige jaren na de implementatie van *APL* werd het nodig geacht gegevens niet alleen uit eigen werkgeheugens of bibliotheken te kopiëren, maar ook uit andere opslagplaatsen van het totale computersysteem. Het *APL*-systeem werd uitgebreid met de *shared variable facility* of de mogelijkheid van gemeenschappelijke variabelen. Het is nu mogelijk gegevens te sturen naar en te ontvangen van andere gebruikers, de gegevensbanken van het grote computersysteem te raadplegen, gegevens die niet in een werkgeheugen passen weg te schrijven naar een gegevensbestand op schijf, en gebruik te maken van andere eenheden dan alleen een terminal die verbonden zijn met de computer.

Het principe van het gebruik van gemeenschappelijke variabelen is heel eenvoudig. De variabele is net zoals alle andere variabelen een lokale of globale variabele, maar de inhoud van deze variabele kan door bijvoorbeeld twee verschillende gebruikers worden gezien. Daartoe moeten deze twee gebruikers een overeenkomst hebben dat deze variabele door beiden mag worden gebruikt. Dit doet men met de systeemfunctie $\square SVO$.

Zodra de variabele is bepaald kan de ene gebruiker de inhoud ervan wijzigen zodat de andere daarna ook met deze inhoud moet werken. Verschillende variabelen mogen op deze manier worden gebruikt, met dien verstande dat er per variabele maar twee gebruikers mogen zijn die over de inhoud kunnen beschikken. Om dit op een goede manier te laten verlopen zijn er naast de systeemfunctie om de variabele aan te vragen nog enkele andere systeemfuncties. In totaal zijn dit er vier, namelijk:

- $\square SVO$ om de variabelen als gemeenschappelijke variabele aan te vragen.
- $\square SVQ$ om te vragen of er gemeenschappelijke variabelen zijn.
- $\square SVC$ om het simultaan gebruik te controleren.
- $\square SVR$ om het simultaan gebruik te beëindigen.

Een paar korte voorbeelden zullen duidelijk maken hoe deze functies kunnen worden gebruikt. We verwijzen de vergevorderde gebruiker voor verdere details naar de literatuurlijst en naar de handleiding van het *APL*-systeem dat gebruikt wordt.

Om de variabele *SIM* met een andere gebruiker te delen, geven we de instructie:

14 $\square SVO$ PRAAT

2

waarbij de naam van de variabele als een karaktervector is gegeven, en het nummer links van de dyadische functie het aansluitingsnummer van de gebruiker is. Dit nummer mag echter ook het nummer zijn van de computer of een bepaalde afdrukeenheid, een magnetische band of een magnetische schijf. Het *APL*-systeem antwoordt of met de code 0 wat betekent dat het niet mogelijk was deze

variabele als een gemeenschappelijke variabele te gebruiken omdat de variabele naam al als naam bestond, of met de code 1 als het gevraagde aansluitingsnummer nog geen antwoord heeft gegeven, of met de code 2 als de variabele simultaan kan worden gebruikt. Diezelfde codes dienen als antwoord bij het vragen naar de status van de variabele zoals in:

2 14 $\square SVQ$ PRAAT

De systeemfunctie wordt in dat geval in de monadische vorm gebruikt.

De status kan echter ook op een andere manier worden gevraagd, namelijk door het gebruik van de systeemfunctie $\square SVQ$ in de monadische vorm als:

$N \square SVQ$

waarbij N een scalaire, een vector met één element, of een lege vector kan zijn. In de eerste twee gevallen is N het nummer van een andere gebruiker of een *processor* verbonden met het computersysteem. De functie geeft dan als uitvoer de namen van de variabelen die deze andere gebruiker of processor heeft aangevraagd als gemeenschappelijke variabelen. In het derde geval worden alleen de nummers van de vragende *processors* of gebruikers weergegeven.

De systeemvariabele $\square SVC$ kan monadisch en dyadisch worden gebruikt. In de monadische vorm vraagt men de toestand aan van de aanvraag of het gebruik van een bepaalde variabele, en in de dyadische vorm wordt de volgorde van gebruik vastgelegd. Voor het gebruik van deze functie in beide vormen refereren we naar de literatuurlijst.

De laatste systeemvariabele $\square SVR$ staat de gebruiker toe een gemeenschappelijke variabele terug te nemen zonder dat men daarvoor van werkgeheugen hoeft te veranderen. Het gebruik van deze functie is heel eenvoudig:

1 $\square SVR$ PRAAT

waarbij de uitvoer de code 0 moet hebben om de variabele terug te nemen. Zonodig moet men de functie enkele malen opnieuw gebruiken.

Voor de systeemfuncties betreffende de gemeenschappelijke variabelen en ook voor de andere systeemfuncties, systeemvariabelen en systeemcommando's raden we de gebruiker aan de referentieboeken behorende bij het *APL*-systeem te raadplegen.

10 Nabeschouwing

We zijn bij dit hoofdstuk aangekomen na een uitgebreide kennismaking met de taal en haar diverse gebruiksmogelijkheden. Er blijft nochtans heel wat te vertellen over programmeren in het algemeen of in *APL* dan we in de hoofdstukken 3 en 8 hebben gedaan. Wanneer men wordt geconfronteerd met een probleem doet men er goed aan zich eerst af te vragen of het noodzakelijk is via automatisering de oplossing te bereiken. Men kan hierbij enkele vragen stellen over het doel en leefbaarheid van een programma. Dient het programma slechts één enkele keer te worden gebruikt, en bereikt men de gewenste resultaten wel zonder dat de gebruiker zelf nog heel wat 'handwerk' dient te doen?

In het algemeen rijst de vraag wat men moet programmeren en wat *niet* moet. Is het bijvoorbeeld nodig het gezinsbudget in een geautomatiseerde vorm bij te houden? Alhoewel we de functies over dit onderwerp als toepassing van enkele technieken hebben geschreven, is het ook hier goed zich af te vragen of men het niet evengoed met de hand kan doen. In vele gevallen berust het antwoord op subjectiviteit van de programmeur of gebruiker, mede door het feit dat zo'n antwoord afhankelijk blijkt te zijn van bepaalde omstandigheden.

Anders ligt het bij de financiële planningsfunctie. Om de verscheidenheid van rapportering toe te laten is zo'n soort functie uiterst geschikt. Er blijft echter een grotere vorm van initiatief nodig om de functie te laten uitvoeren. De gebruiker moet namelijk het rapport beschrijven, inclusief vormgeving en berekeningen, zodat men de functie *FINAN* kan beschouwen als een *pre-processor* die de instructies omzet naar *APL*. De invoer is in dit geval een vorm van programmeren die aan een bepaalde syntax moet beantwoorden.

Een andere vraag die moet worden gesteld betreft de vorm van een te ontwerpen functie. Onder vorm verstaan we hier zowel de keuze van hulpfuncties, de volgorde van invoer, instructies en uitvoer. Omdat de taal in dit boek voorop stond zijn we niet ingegaan op deze problematiek. Dit neemt echter niet weg dat het begrip vorm een niet-triviaal iets is in de context van probleemoplossen. In de meeste gevallen hebben we te maken met wat men soms een *black box* noemt. Deze box is de gedefinieerde *APL*-functie of in het algemeen een computerprogramma.

De *black box* kan worden beschreven aan de hand van drie onderdelen, namelijk:

1. Een invoergedeelte (gegevens naar de box).

2. Bewerkingsgedeelte (het gebeuren in de box).
3. Een uitvoergedeelte (resultaten naar de gebruiker toe).

De gebruiker van een functie of programma is meestal alleen maar geïnteresseerd in de invoer en de uitvoer. De analist of programmeur daarentegen heeft tot taak een probleem op te lossen door het invullen van het bewerkingsgedeelte, er mee rekening houdende dat verschillende vormen van invoer mogelijk zijn en dat de uitvoer op een bepaalde manier dient te worden gegeven. Wie in *APL* functies ontwerpt krijgt dus met alle onderdelen van de black box te maken.

Afhankelijk van de graad van gebruiksvriendelijkheid van de functie zal de vraag naar invoer op een bepaalde (interactieve) manier verlopen. De bewerkingen, of het hart van de functie, duren liefst niet te lang zonder onderbreking en een regelmatig contact met de gebruiker is derhalve gewenst. *APL* is immers een terminal-gerichte taal. Maak het ook de gebruiker niet moeilijk met instructies die beschrijven hoe het gedane werk in een bibliotheek moet worden bewaard. Stel een vraag of het werk voor later gebruik nodig is en voer als het kan de systeem-commando's binnen de functie uit.

Dikwijls zal men voor de paradox komen te staan waarbij het gemak voor de gebruiker ten koste gaat van de moeilijkheid voor de programmeur. Elke aanwijziging of vuistregel is hier verder overbodig zolang men tijdens de analyse en het programmeren van een functie de gebruiker in gedachten houdt. Kijkt men alleen maar naar probleemoplossen op zich dan zijn er nog enkele opmerkingen te maken.

Programmeren is niet alleen het samenstellen van een verzameling van technieken. Men gaat zo veel mogelijk gestructureerd aan het werk. Op die manier berust het programmeren op *common sense*. Men mag immers niet vergeten dat programmeren een activiteit is waar men eigenlijk nooit de kleinste details over het hoofd kan zien. Programmeren bestaat meer uit het samenvoegen van kleine delen van instructies tot hulpfuncties en hoofdfuncties. Dit moet gebeuren in de context van een grotere structuur die als doel heeft een probleem op te lossen. En toch bestaat het gevaar dat de kleinere gedeelten na verloop van tijd geen aandacht meer krijgen. Er bestaat zelfs een grote kans dat men na enige tijd niet meer weet wat er nou eigenlijk in een hulpfunctie gebeurt, of erger nog, wat de betekenis van een variabele is, of waarom men naar een bepaalde regelnaam springt. Is men een detail vergeten en moet het alsnog 'ergens' tussen komen te staan dan kan men zich ergernis besparen door de functies op een gestructureerde en gedocumenteerde manier te ontwerpen.

Dit soort problemen is algemeen omdat programmeurs veelal programmeren met de gedachte dat ze bepaalde instructies toch maar één keer schrijven en dat wanneer een functie werkt, men er niet meer naar om hoeft te kijken. Het tegenovergestelde lijkt waar te zijn. Na het schrijven van een reeks functies of tijdens de uitvoering ervan bemerkt men heel dikwijls dat het juist die kleine vergeten details zijn die het terug in een functie duiken moeilijk maken.

Gestructureerd programmeren is een kreet die soms schouderophalend wordt afgewezen. Men vergeet hierbij soms dat het een naam is voor een techniek waarbij de programmeur het gemakkelijker heeft en meer betrouwbare programmatuur aflevert. In dit boek hebben we bijvoorbeeld geen enkele *flow chart* gebruikt. Wellicht zouden enkele functies vlugger begrepen zijn wanneer we dit wel gedaan hadden, maar dan zou dit ten koste zijn gegaan van bladzijden vol beschrijvingen in rechthoekjes aan elkaar gekoppeld met lijntjes en pijlen, dit alles gewrongen tussen een begin en een einde.

Het lijkt ons veel beter een structuurkaart samen te stellen waarbij in een duidelijke omschrijvende taal per regel wordt aangeduid wat er dient te gebeuren. Komt het voor dat een test wordt uitgevoerd dan kan men de volgende regel na de beschrijving van de test laten volgen door enkele naast elkaar staande beschrijvingen, één voor elke verwijzing die kan voorkomen. Op die manier leest men van boven naar onder een volledig verhaal over de gehele functie of delen ervan.

Zeer belangrijk is het dat de programma's kort zijn en dat indien nodig gebruik wordt gemaakt van hulpfuncties die een bepaald doel hebben. Een *APL*-functie zou in feite niet langer hoeven te zijn dan 25 regels, enige uitzonderingen daargelaten. Vergeet in elk geval niet dat hoe langer een functie is, hoe meer fouten er kunnen voorkomen en hoe moeilijker het wordt ze op te sporen. Vooral bij het later invoegen van vergeten details wordt een lange functie een kluwen van verspringen naar regels die uitzonderlijke toestanden in het probleemoplossen verwerken. Op dat moment doet men er beter aan de functie opnieuw te ontwerpen en te programmeren.

Nog een belangrijk element bij gestructureerd programmeren is het gebruik van variabelenamen met een betekenis of een beschrijvend karakter. Het gebruik van abstracte namen zoals ABC1 of YWE maakt het inderdaad moeilijk de functies in een later stadium te lezen, laat staan te interpreteren. Aan de andere kant is de keuze van een variabele naam soms persoonlijk en kunnen misverstanden rijzen. Een variabele met de naam NUL bijvoorbeeld die de waarde één kan bevatten is geen variabele: Nul is een constante. Soms gebeurt het dat de gebruikte naam meerdere betekenissen kan hebben of voor verschillende doeleinden in een functie wordt gebruikt. We raden aan de namen ook niet te lang te maken. Vele *APL*-systemen houden toch maar een aantal karakters per naam in het geheugen!

Samengevat kan men stellen dat de *common sense* regels nog de beste blijken te zijn. Functies moeten leesbaar blijven zowel voor de programmeur zelf als voor anderen. Een goed ontworpen structuur is meteen een beschrijving van een functie. Alhoewel er in dit boek geen gebruik van is gemaakt, doet men er goed aan regelmatig een commentaarregel in te voegen, desnoods met een verwijzing naar de structuurkaart.

Probleemoplossen met *APL* komt voor een groot deel neer op het maken van een goed ontwerp. Een goed boek over hoe men problemen analyseert en de te programmeren functies ontwerpt is echter niet noodzakelijk een *APL*-boek, omdat het een algemeen probleem betreft. Het ontwerp helpt de complexiteit van het programmeren te reduceren zodat betere programma's of functies kunnen worden geschreven. Pas als men gestructureerd programmeren goed onder de knie heeft, kan men gaan praten over andere factoren zoals het herschrijven van instructies opdat de verwerkingen sneller kunnen gaan, minder plaats in het werkgeheugen wordt ingenomen, enz. Een goed ontwerp laat ook op een vlotte manier toe latere aanpassingen te maken. Programmeren in *APL* blijft echter een complexe en intellectuele activiteit en verdient een systematische en structurele aanpak om het succes van de toegepaste programmatuur te verzekeren.

Appendix A

Foutmeldingen

Melding	Betekenis en eventuele verbetering.
<i>ALREADY SIGNED ON</i>	Terminal is reeds in gebruik – sluit de sessie af en maak een nieuwe aansluiting.
<i>CHARACTER ERROR</i>	Onbekend karakter, waarschijnlijk verkeerde samenstelling – type het correcte karakter in.
<i>DEFN ERROR</i>	Verkeerd gebruik tijdens definitie mode of verbetering van een functie omdat de functie reeds geopend is – kijk de definitie regel na of sluit eerst de functie.
<i>DEPTH ERROR</i>	Er zijn te veel niveaus in functies gebruik makende van hulpfuncties.
<i>DOMAIN ERROR</i>	Een operator of functie is gebruikt met verkeerde linkse of rechtse gegevens – kijk de vorm of dimensie na van die gegevens.
<i>IMPLICIT</i>	Niet toegelaten waarde of geen waarde gegeven aan de systeemvariabele.
<i>IMPROPER LIBRARY REFERENCE</i>	De naam van het werkgeheugen mag niet worden gebruikt voor een)SAVE – wijzig de naam met een)WSID. Het is ook mogelijk dat het werkgeheugen niet bestaat bij een)LOAD.
<i>INCORRECT SIGN-ON</i>	De aansluitingsprocedure is fout – probeer opnieuw of raadpleeg de gebruikersgids bij het APL-systeem.
<i>INDEX ERROR</i>	Een referentie werd gemaakt naar een element van een vektor of matrix dat buiten de lengte of dimensie valt – kijk de indexwaarde of -variabele na.

INTERFACE QUOTA EXHAUSTED	Er worden meer gemeenschappelijke variabelen gebruikt dan toegelaten – vraag om uitbreiding aan de verantwoordelijke voor APL.
INTERRUPT	De uitvoering van een functie werd onderbroken door een gebruiker terwijl het systeem invoer verwachtte.
LENGTH ERROR	Er is geprobeerd vektoren of matrices met elkaar te verwerken terwijl deze van ongelijke lengte of vorm zijn – pas de lengte of de vorm aan of gebruik een andere APL-functie.
MESSAGE LOST	Een onderbreking of storing heeft plaatsgevonden voor het bericht is aangekomen – probeer opnieuw.
NO SHARES	De mogelijkheid tot het gebruik van gemeenschappelijke variabelen is niet aanwezig op het APL-systeem of onder uw aansluitnummer.
NOT COPIED:	De elementen in de lijst komen niet voor in het werkgeheugen.
NOT ERASED:	De elementen in de lijst kunnen niet worden verwijderd – zeer waarschijnlijk zijn dit onderbroken functies – maak de statuslijst vrij.
NOT FOUND:	De elementen in de lijst werden niet gekopieerd bij)PCOPY omdat ze reeds voorkwamen in het actief werkgeheugen.
NOT GROUPED, NAME IN USE	De gebruikte groepnaam is niet toegelaten daar deze reeds werd gebruikt als variabele- of functienaam.
NOT SAVED, THIS IS	Het werkgeheugen kan niet worden weggeschreven omdat het geen naam heeft of uit een publieke bibliotheek komt – geef het werkgeheugen een nieuwe naam.
NOT SAVED, WS QUOTA USED UP	Het maximaal aantal toegelaten werkgeheugens onder dit aansluitnummer werd reeds bereikt – schrijf de inhoud van het actief werkgeheugen naar)CONTINUE of in andere werkgeheugens.

*NOT WITH OPEN
DEFINITION*

De instructie niet uitgevoerd is niet toegelaten terwijl een functie geopend is – sluit eerst de functie en probeer dan opnieuw.

NUMBER IN USE

Iemand anders gebruikt reeds het APL-systeem met dit aansluitnummer – kijk dit nummer na of ga na wie het nummer gebruikt.

NUMBER LOCKED OUT

Het aansluitnummer is onbruikbaar gemaakt – raadpleeg de verantwoordelijke voor APL.

NUMBER NOT IN SYSTEM

Het nummer is geen APL aansluitnummer of een verkeerd wachtwoord werd gebruikt – kijk het nummer en/of het wachtwoord na. Indien het wachtwoord vergeten is, raadpleeg de verantwoordelijke voor APL.

RANK ERROR

De structuur van het gegeven is fout.

SI DAMAGE

Een onderbroken functie heeft verandering ondergaan op de regel met de functienaam of op een regel met een verwijzingsnaam, of de functie is reeds verwijderd. Deze melding kan ook voorkomen als een onderbroken functie wordt gewijzigd en niet bovenaan de statuslijst voorkwam.

SYMBOL TABLE FULL

Er zijn te veel verschillende namen in het werkgeheugen – vergroot de tabel aan de hand van het)SYMBOLS commando.

SYNTAX ERROR

Ontoelaatbare syntax in de instructie. Het is mogelijk dat een gebruikte naam te veel of te weinig gegevens heeft.

SYSTEM ERROR

Een fout is gevonden welke niet door APL kan worden opgevangen. Het is mogelijk dat op dit moment alle gegevens uit het actief werkgeheugen verwijderd zijn – raadpleeg onmiddellijk de verantwoordelijke voor APL.

VALUE ERROR

De variabele heeft geen waarde.

WS FULL

Het werkgeheugen is vol – verwijder namen van gegevens of functies die niet meer nodig zijn.

WS LOCKED

Het werkgeheugen heeft een wachtwoord en geen of het verkeerde wachtwoord werd opgegeven.

WS NOT FOUND

Een werkgeheugen onder de gegeven naam bestaat niet.

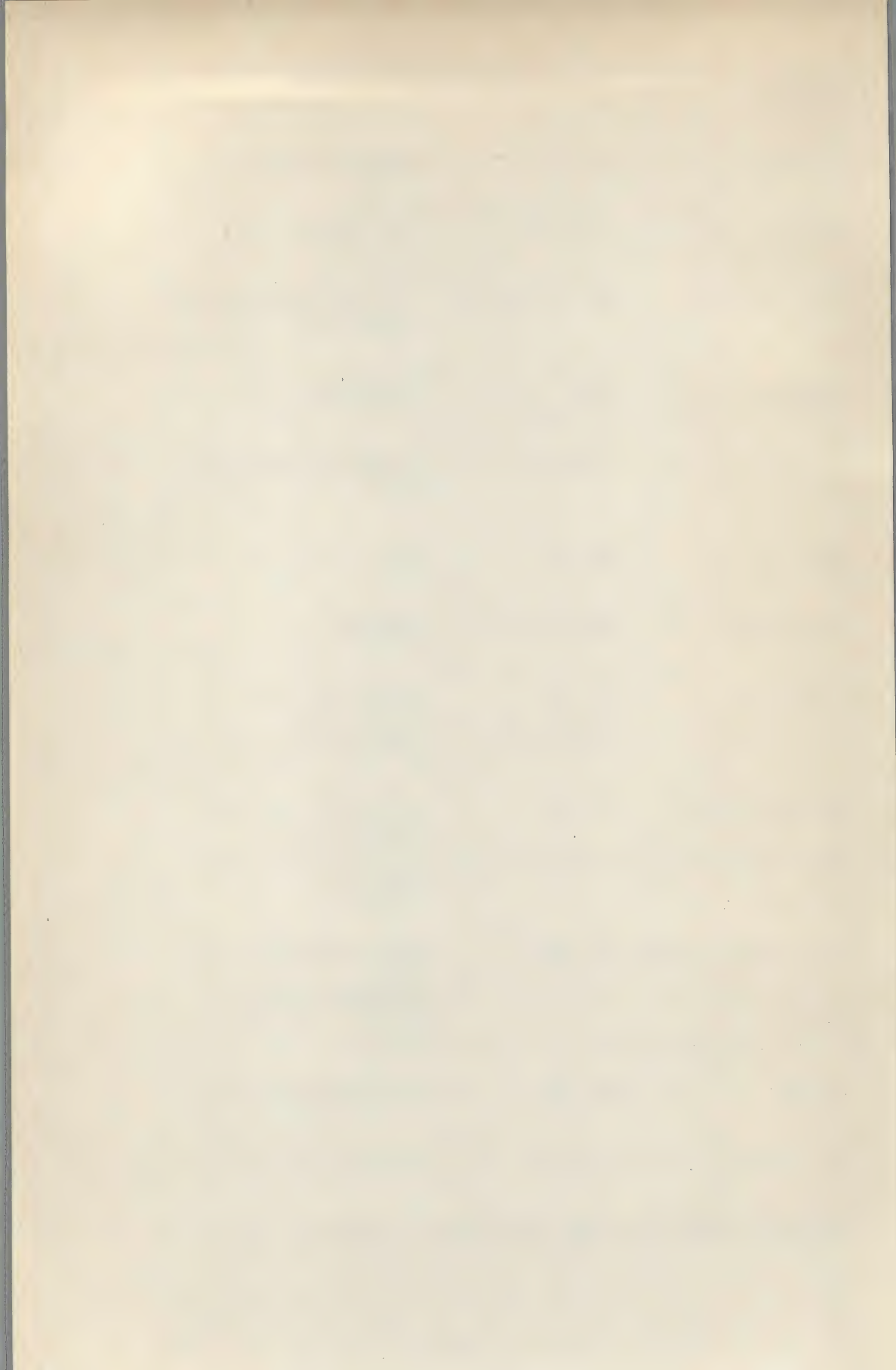
Appendix B

Beschikbare APL-systemen, 1983*

<i>Leveranciers:</i>	<i>APL versie</i>	<i>Beschikbaar op/ onder:</i>
Burroughs	APL/700	B6000/B7000-series
Control Data	APL*Cyber APL2	Cyber-serie Cybernet-service
Data General Corp.	AOS/VS APL	DGC Eclipse MV/6000 en MV/8000
Digital	APLSF APL-II (obsolete) VAX-II APL	DEC 10 en DEC 20 PDP 11 serie VAX serie
GEC	4000 APL	GEC 4000 serie
Harris	Harris APL	systemen onder VULCAN OS
Hewlett Packard	APL/3000	HP 3000-III serie
Honeywell	APL/64, APL/66	Level 64, Level 66 serie
IBM	VSAPL DPPX APL APLSV-subset APL2 (experimenteel)	370-serie (vanaf /138) 4300-serie 303x/308x-serie 8100 serie 5100 serie systemen met VM/370- CMS

IP Sharp	IPSA	IBM en IBM-PCM
ICL	2900 APL	ICL 2900 serie
OCM, Quebec	APL*MYRIADE	Texas Instruments 990/10 en 990/12 serie
Siemens	APL1	System 4004 en 7000
Sigma AMS/IPL	VIZ::APL	Op Z80-CP/M gebaseerde micro's
Sems	APL/16	Solar serie
Sperry Univac	APL/1100	1100-serie
STSC	APL*Plus APL*Plus/80 APL*Plus/PC	IBM en IBM-PCM Tandy TRS-80 IBM PC/XT
Tel. Integr. Systems	TIS-APL	Op sommige CP/M geba- seerde micro's
Waterloo Comp. Systems	Waterloo Micro APL	Op sommige CP/M geba- seerde micro's
The Computer Company	APL. 68000	Spectrum, SAGE/Scimitar, Wicat Op MC68000 gebaseerde micro's
Vanguard	APL/V80	Op Z80 gebaseerde micro's
Xerox/Atkins	Xerox APL	Xerox 560, Sigma 6/7/9

* Met dank aan dhr. Theo Zwart, BSO/Management Support b.v., Utrecht.



Register

- AND functie $\wedge$ 49
- aansluiting
 - informatie $\square AI$ 195
 - nummer 189
- absolute waarde | 45
- actief werkgeheugen 183
- afronden 42
- afsluiten 190
- alfabetisch(e)
 - karakters 26
 - vector 26
- alfanumerieke gegevens 26, 136
- algoritme 54
 - product 89
 - som 89
- Atomic Vector $\square AV$ 22, 195
- ATTENTION-toets
 - onderbreken functie 133, 152
 - verbetering fout 21, 65
 - verwijderen functieregel 65, 146
- BACKSPACE-toets 21
- basis logaritme 126
- berichten
 - gebruiker 188
 - operator 187
 - systeem 188
- bibliotheek
 - gebruiker 184, 188, 189
 - publieke 33, 188
- binaire vector 70, 86
- bit 19, 20
- budgettering 166
- byte 19
- calculator manier 23
- canonische voorstelling $\square CR$ 198
- combinatiefunctie ! 124
- commando
 -)CLEAR 186
 -)CONTINUE 185, 190
 -)COPY 33, 186, 198
 -)DIGITS 135
 -)DROP 186
 -)ERASE 72, 191, 199
 -)FNS 191
 -)GROUP 191
 -)GRP 191
 -)GRPS 191
 -)LIB 184
 -)LOAD 184
 -)MSG 188
 -)MSGN 188
 -)OFF 190
 -)OFF HOLD 190
 -)OPR 187
 -)OPRN 187
 -)ORIGIN 81
 -)PCOPY 186
 -)PORTS 187
 -)SAVE 28, 189
 -)SI 148–150
 -)SIV 192
 -)SYMBOLS 193
 -)VARS 191, 192
 -)WIDTH 193
 -)WSID 32, 184
- commentaartekens $\textcircled{R}$ 102, 128
- communicatie
 - terminal en computer 185–188
 - werkgeheugen en computer 202
 - werkgeheugen en gegevensbanken 202
- component, van een vector 30

- computer
 - main-frame 17
 - micro 19
 - mini 17
- conditionele sprong 69, 74, 90
- cosinus 128
- datum 178, 180, 198
- decimaal 23
- Del Δ 154
- Delay $\square DL$ 201
- dubbele punt
 - bij verwijzingsplaats 74
 - met $\square$ als invoer 60
 - met *password* 190
- dyadische functie 39
- element van $\in$ 43, 49
- executie 141
- expliciet resultaat 59
- exponent 41
- exponent-teken $\times$ 41
- exponentiële vorm 25
- faculteitsfunctie ! 124
- formaat
 - dyadisch Φ 136–137, 139
 - monadisch Φ 138
- fout
 - in een uitdrukking 33
 - in gedefinieerde functie 147, 149
 - rapportering (zie appendix A) 33, 34, 43, 77, 81, 82, 93, 133, 150, 151, 192, 194
 - verbetering 21
- functie
 - afsluiten 56, 64
 - gedefinieerd 54
 - lijst van *JFNS* 191
 - onderbreken 148, 152
 - primitieven 38
 - types van 58
 - verzegelen 145
- functieaanmaak $\square FX$ 199
- functienaam 55
- Gamma functie 124
- gebruikerslijst $\square UL$ 198
- gedefinieerde functie
 - met expliciet resultaat 59
 - met impliciet resultaat 59
 - met lokale variabelen 69
 - naamgeving 55
 - verbetering in 66
- geheugen
 - flexibel 20
 - vast 20
- gemeenschappelijke variabelen 202
- gemiddelde 172
- globale variabele 60, 71
- goniometrische functies 117, 125
- groter dan $>$ 50
- impliciet resultaat 54
- index
 - generator $\vdash$ 42–43
 - oorsprong $\square IO$ 81, 194
- indicering
 - van een vector 33, 80, 81
 - van een matrix 101
- inversie van matrix $\boxminus$ 112
- invoer 60, 61, 131
- inwendig product 94, 110
- karakter
 - als data 61
 - als gegeven gedefinieerde functie 161
 - vector 26, 27
- kleiner dan $<$ 50
- kleiner of gelijk aan $\leq$ 50
- komma
 - als samenvoeging 30
 - niet in decimale getallen 30
- latente uitdrukking $\square LX$ 196
- lengte van een vector 46
- lineaire vergelijkingen 87, 113
- lokale variabele 71, 72
- logaritmefunctie $\otimes$ 126
- logische functies

- element van $\in$ 43, 49
- AND $\wedge$ 49
- NAND ∇ 50
- NOR ∇ 50
- OR $\vee$ 49
- gelijk aan $=$ 50
- groter dan $>$ 50
- groter of gelijk aan $\geq$ 50
- kleiner dan $<$ 50
- kleiner of gelijk aan $\leq$ 50
- niet gelijk aan $\neq$ 50
- ontkenning (NOT) $\sim$ 50
- matrix
 - definitie 47, 98
 - deling 113
 - inversie 113
 - ontbinden 48
 - produkt 110–111
 - samenvoegen 48
- maximum $\lceil$ 42
- minimum $\lfloor$ 42
- monadische functie 39
- NAND functie ∇ 50
- NOR functie ∇ 50
- naam
 - gemeenschappelijke variabelen 202
 - lijst $\square NL$ 200
 - regel 74
 - type $\square NC$ 200
 - variabelen 28
 - werkgeheugen 32
- OR functie 50
- onderlijnen 128
- ontkenning (NOT) $\sim$ 50
- primitieve functies
 - AND $\wedge$ 49
 - NAND ∇ 50
 - NOR ∇ 50
 - OR $\vee$ 49
 - afronding naar beneden $\lfloor$ 42
 - afronding naar boven $\lceil$ 42
 - afrekken $-$ 23
 - alfanumerieke invoer $\square$ 133
 - coderen $\top$ 87, 88
 - combinatorische $-$! 124
 - cosinus 128
 - decoderen $\perp$ 86
 - delen door $\div$ 43
 - element van $\in$ 43, 49
 - exponentie $\times$ 41
 - formaat Φ 136
 - gelijk aan $=$ 50
 - groter dan $>$ 50
 - groter of gelijk aan $\geq$ 50
 - index generator $\wr$ 42–43
 - kleiner dan $<$ 50
 - kleiner of gelijk aan $\leq$ 50
 - logaritme $\otimes$ 126
 - machtsverheffing $\times$ 41
 - matrix deling $\boxdiv$ 113
 - maximum $\lceil$ 42
 - minimum $\lfloor$ 42
 - niet gelijk aan $\neq$ 50
 - numerieke invoer $\square$ 69
 - omkering $\ominus$ 107
 - ontkenning (NOT) $\sim$ 50
 - ontrafelen 48
 - opnemen $\uparrow$ 104
 - pi 125
 - plus $+$ 23
 - random $?$ 46
 - reciproke $\div$ 40
 - reductie $/$ 88, 109
 - restant $|$ 45
 - rotatie $\oplus$ 84, 105
 - samenvoeging $,$ 48
 - sinus 128
 - sorteren hoog naar laag ∇ 85
 - sorteren laag naar hoog $\Uparrow$ 85, 108
 - tangens 129
 - teken van een getal $\times$ 40
 - transponeren $\oslash$ 107
 - uitbreiding $\setminus$ 91, 108
 - uitvoering Φ 136
 - vermenigvuldigen $\times$ 33
 - vormgeving ρ 46, 47, 62

- weglaten ↓ 83
- quad □ 131
- quad quote □ 132, 140–141
- RESEND 33
- reductie 56
- regelteller □LC 196
- restwaarde 45
- RETURN-toets 21
- samenvoeging
 - scalair 30, 62
 - vectoren 62, 134
- scalaire 39
- sinus 128
- sorteren
 - hoog naar laag ↕ 85
 - laag naar hoog ↕ 85, 108
- systeemfuncties
 - share control* □SVC 203
 - share offer* □SVO 202
 - share query* □SVQ 203
 - share retract* □SVR 203
 - canonische functie □CR 198
 - delay* functie □DL 201
 - dynamisch verwijderen □EX 199
 - functieaanmaak □FX 199
 - naamlijst □NL 200
 - naamtype □NC 200
- systeemvariabelen
 - atomic vector* □AV 22, 195
 - breedte uitvoer □PW 193
 - index oorsprong □IO 81, 194
 - informatie aansluiting □AI 195
 - latente uitdrukking □LX 196
 - lijst gebruikers □UL 198
 - precisie afdruk □PP 153
 - random *link* □RL 197
 - regelteller □LC 196
 - terminal type □LC 198
 - tijd en datum functie □TS 198
 - tolerantie vergelijking □CT 195
 - vrij werkgeheugen □WA 198
 - tijd en datum variabele □RS 198
 - toetsen (logisch) 67
 - toetsenbord 20, 21
 - toevalsgetallen 44
 - trace 154
 - transpositie 84, 105–106
 - uitbreiding \ 91, 109
 - uitvoer
 - dyadisch 136
 - gedefinieerde functie 59
 - uitvoering ⊕ 136
 - uitwendig product 93, 110
 - variabele
 - gemeenschappelijk gebruik 202
 - globaal 60, 71
 - lijst 200
 - lokaal 71
 - namen 28
 - vector
 - bewerkingen op – 82
 - definitie 39
 - dimensies 39, 46, 47
 - nul of leeg 46
 - vector 38
 - vergelijkingstolerantie □CT 195
 - verwijdering
 - van functie met □EX 199
 - van functie met)ERASE 191
 - van regel in een functie 65, 146
 - van werkgeheugen 186
 - verwijzingspijl
 - eenvoudig 70
 - conditioneel 70
 - verzegelde functie 145
 - voorbeelden
 - budgettering 166–172
 - computer ondersteund onderwijs 173–175
 - driehoeksmeting 128
 - financiële planning 175–180
 - geldwisselaar 55
 - histogrammen 158–160

omzetberekening in een restaurant

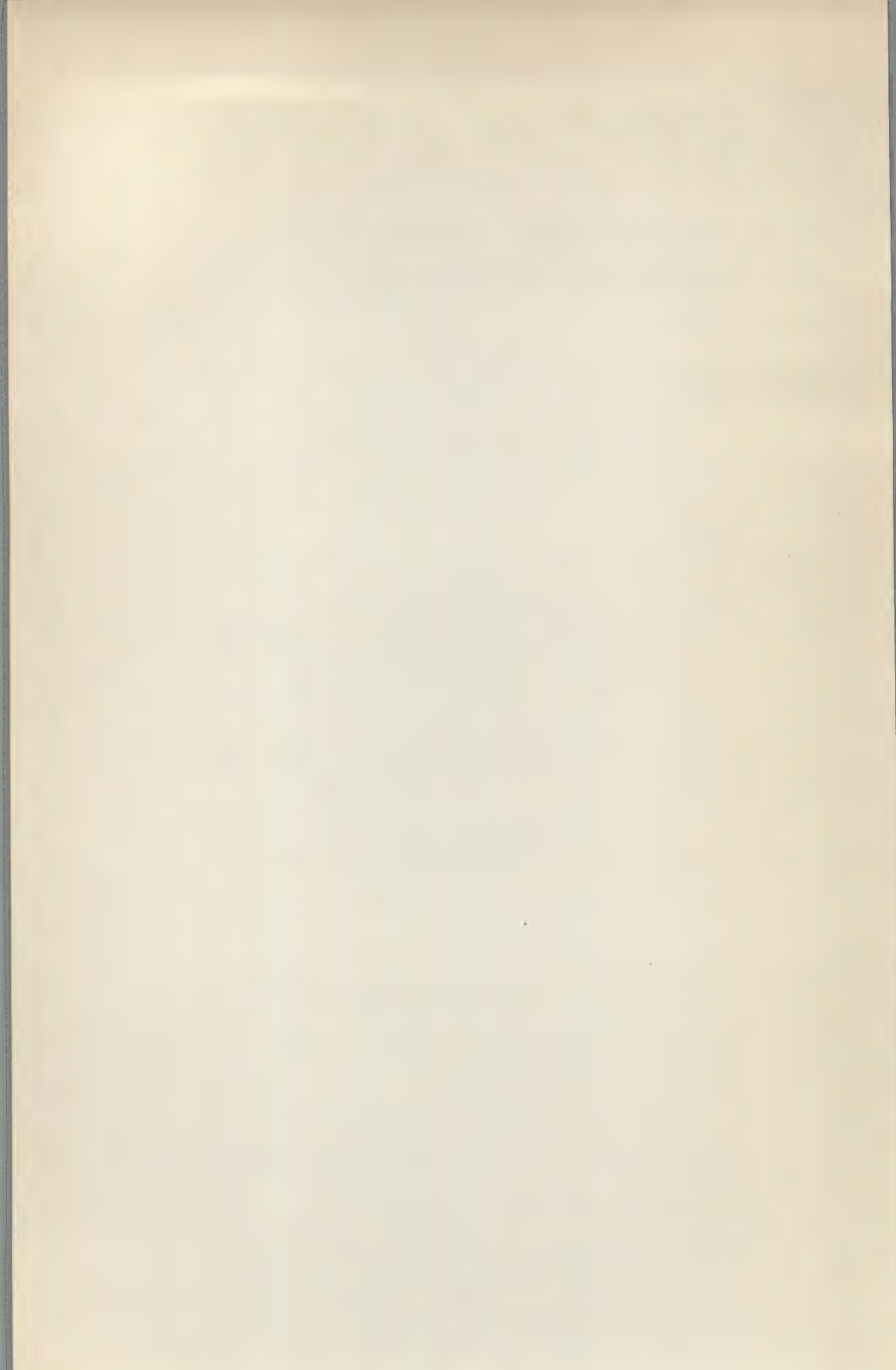
56, 57

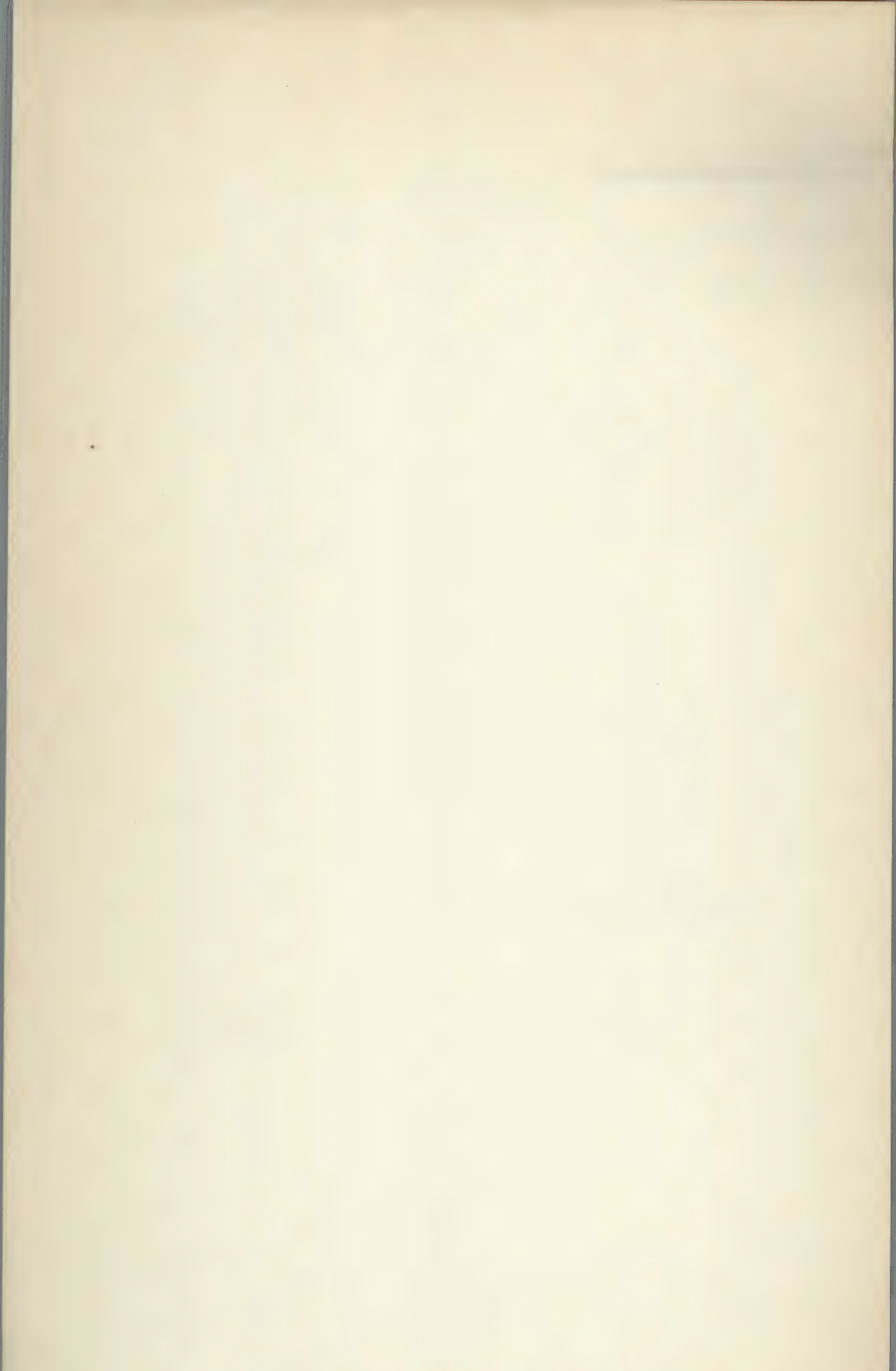
schetsen (grafiek) 118–123

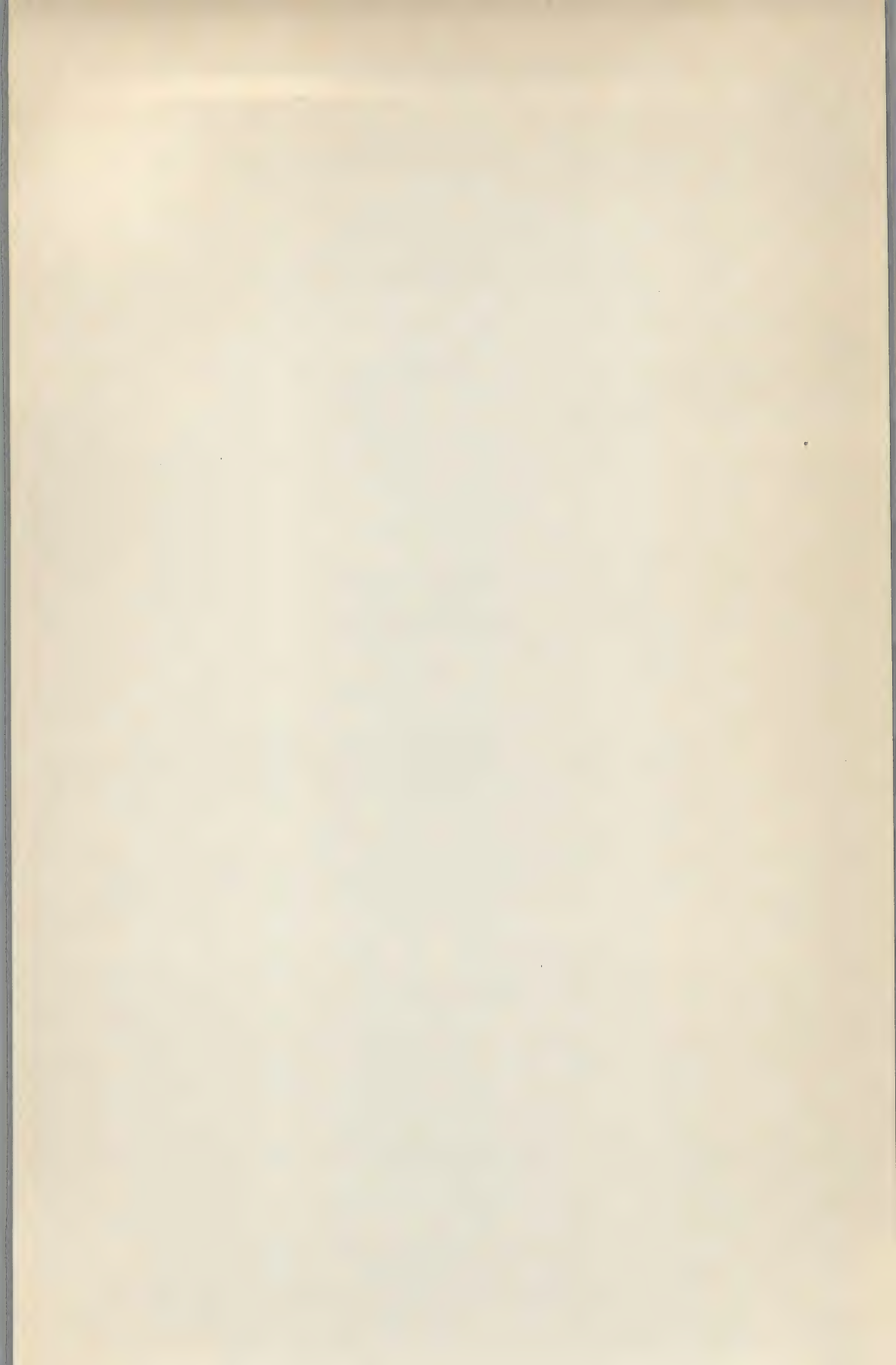
statistiek 172–173

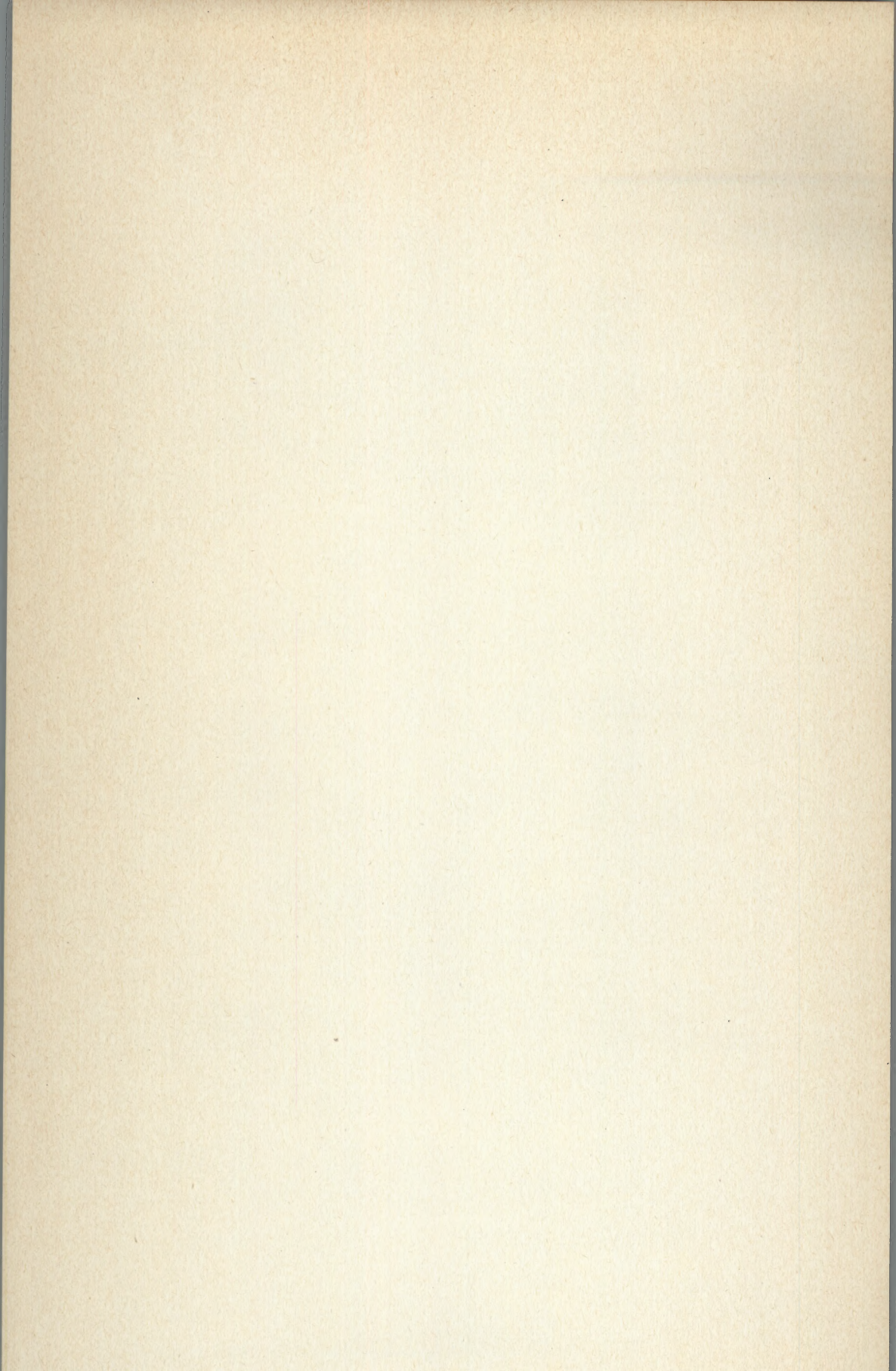
tekstverwerking 161–166

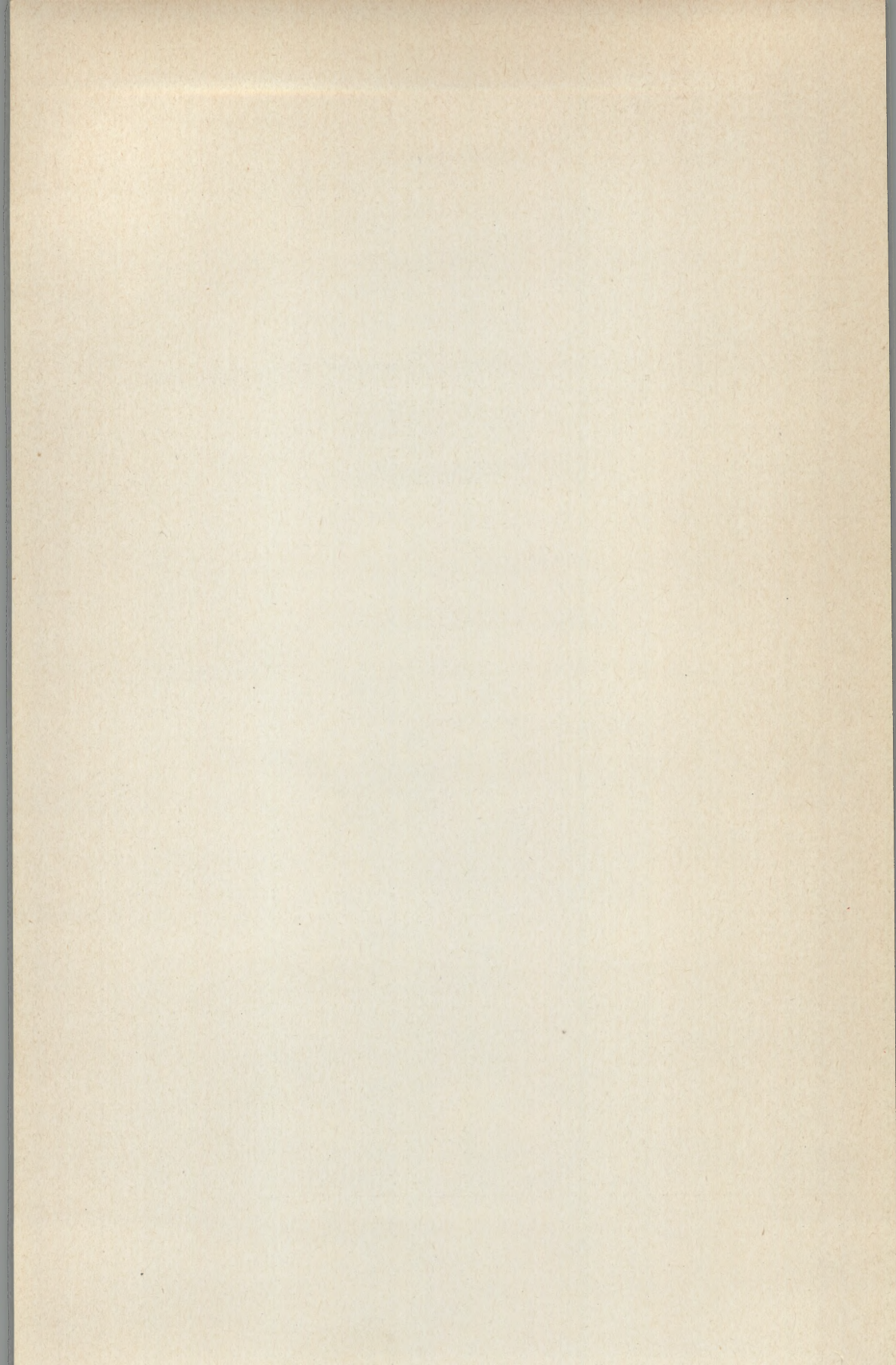
werkgeheugen 32, 183

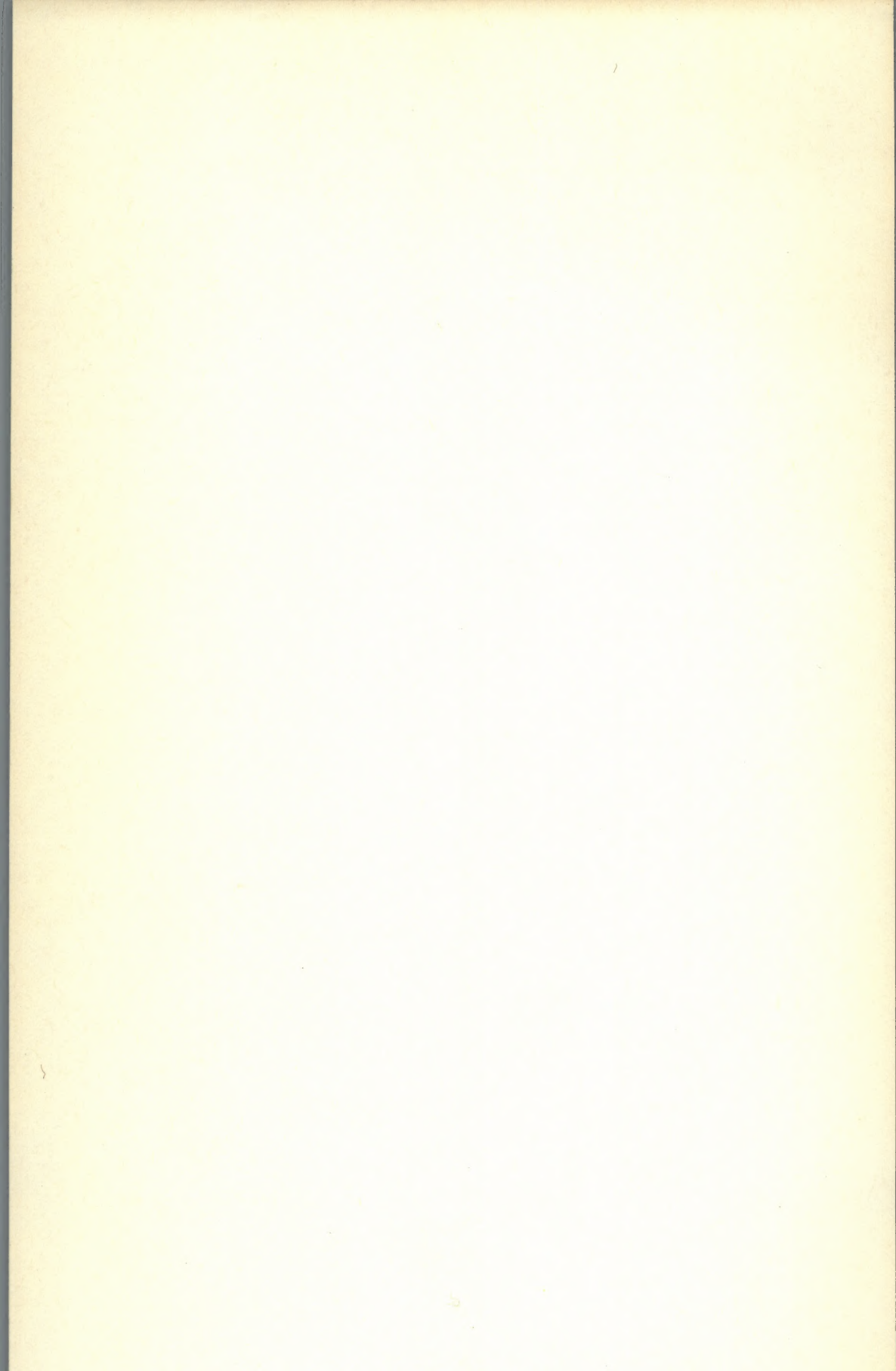












'A Programming Language' ofwel APL is een zeer krachtige computertaal die reeds begin jaren zestig is ontworpen door K. Iverson. Op dit moment beleeft de taal een opleving, niet in het minst door toepassing op microcomputers. Dit is mogelijk vanwege de steeds grotere interne geheugens van micro's. Een APL-interpreter vergt immers nogal wat geheugen.

De kracht van APL mag blijken uit het feit dat men met veel minder instructies een probleem kan oplossen dan in de meeste andere computertalen, zoals bijvoorbeeld BASIC.

APL kent een eigen notatiewijze met speciale karakters die sterk afwijken van andere talen. Vooral het werken met vectoren en matrices is met APL veel eenvoudiger dan met een andere taal. Aanvankelijk werd APL gebruikt door wiskundigen en technici, maar steeds meer ziet men juist in bedrijfsmatige toepassingen het nut in van APL. Ook als ontwerptaal biedt APL veel mogelijkheden.

Dit boek leert de beginnende gebruiker de vele mogelijkheden van APL. Met honderden voorbeelden is het een duidelijk instructieboek geworden. Daarnaast een waardevol naslagwerk voor de APL-programmeur.

De schrijver

Dr. Hugo J.J. Uyttenhove genoot zijn opleiding in de Verenigde Staten waar hij ook promoveerde. Momenteel is hij directeur van het Eindhovense adviesbureau Computing & Systems Consultants. Daarnaast doceert hij aan de TH Eindhoven afdeling Bedrijfskunde. Tevens vervult hij bestuursfuncties; o.m. secretaris Systeemgroep Nederland en bestuurslid APL-werkgroep (sectie Systeemprogrammatuur NGI).